

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**



**FEUP**

# **Digital Archives: Comparative Study and Interoperability Framework**

**Filipe de Lemos**

Report of Project/Dissertation

Master in Informatics and Computing Engineering

Supervisor: Gabriel de Sousa Torcato David (Associate Professor)

2008, July



# **Digital Archives: Comparative Study and Interoperability Framework**

**Filipe de Lemos**

Report of Project/Dissertation  
Master in Informatics and Computing Engineering

Approved in public presentation by the jury:

Chair: Rosaldo José Fernandes Rossetti (Invited Assistant Professor)

---

Main Examiner: José Carlos Ramalho (Assistant Professor)

Examiner: Gabriel de Sousa Torcato David (Associate Professor)

15<sup>th</sup> July, 2008



# Abstract

There is an increasing interest on digital archives for the preservation of digital documents. The main software infrastructure of a digital archive is its repository, which stores the digital objects. Repositories can be created by deploying repository systems.

This project was run in the context of Gisa, a software application developed by ParadigmaXis for the management and description of the content of archives. Since in digital archives the content is digital, it is possible to couple it more closely with software applications. The objective of this project was to perform a comparative study on repository systems, and to develop an extensible module for the integration of Gisa (and other applications) with repositories systems.

In the first phase, the comparative study was performed on three open source repository systems: Fedora; DSpace; and EPrints, over the aspects: interoperability; distribution; scalability; design; security; and UI configuration. Additionally, the commercial product DigiTool was included in the interoperability section of the study, which listed the exposed APIs and standard protocols implemented per functionality. This study set the stage for the development of the interoperability framework. It is based on a hierarchy of adapters, with a common generic interface to the contents of repositories, from which less abstract adapters are derived to implement protocols to access these contents in specific repositories. The framework was implemented as a C# library to be used by applications, and includes a univocal and homogenous identifier scheme and search functionalities that allow a query to be run on multiple repositories simultaneously. Adapters were implemented for repositories exposing OAI-PMH; for repositories based on Fedora; and for simple files viewed as a repository. Applications were developed using the framework and include an OAI-PMH harvester and a repository browser, which uses the generic interface of the framework and can therefore be used to browse any repository with the correspondent adapter.



# Resumo

Existe um aumento no interesse em arquivos digitais na preservação de documentos digitais. A principal infraestrutura de software de um arquivo digital é o repositório, que armazena objectos digitais. Repositórios podem ser criados com base em sistemas para repositórios.

Este projecto decorreu no âmbito do Gisa, uma aplicação de software desenvolvida pela ParadigmaXis para a gestão e descrição dos conteúdos de arquivos. Como em arquivos digitais o conteúdo é por definição digital, é possível ligar este mais estreitamente a outras aplicações de software. O objectivo deste projecto foi efectuar um estudo comparativo de sistemas para repositórios, e desenvolver um módulo para a integração de repositórios com o Gisa e outras aplicações.

Na primeira fase foi feito o estudo comparativo em três sistemas *open source* para repositórios: Fedora; DSpace; e EPrints, sobre os aspectos de interoperabilidade; de distribuição; escalabilidade; *design*; segurança; e configuração do UI. Um produto comercial, DigiTool, foi também incluído no estudo, na secção de interoperabilidade, que lista por funcionalidades as APIs expostas e protocolos normalizados implementados. Este estudo criou as condições para o desenvolvimento de uma *framework* de interoperabilidade. Esta é baseada numa hierarquia de adaptadores, com uma interface genérica comum, da qual são derivados adaptadores menos abstractos que implementam protocolos para o acesso aos conteúdos de um repositório específico. A *framework* foi implementada como uma biblioteca de C# para ser usada por aplicações, e inclui um esquema de identificação unívoca de objectos digitais e integra funcionalidades de pesquisa que permitem interrogar vários repositórios simultaneamente. Foram implementados adaptadores para repositórios com OAI-PMH; para repositórios baseados em Fedora; e para ficheiros simples vistos como sendo um repositório. Foram desenvolvidas aplicações usando a *framework*, incluindo um *harvester* para OAI-PMH e um *browser* de repositórios que utiliza a interface genérica e portanto pode ser usada com qualquer repositório com um adaptador.





# Acknowledgements

I would like to thank Eng. Filipe Correia and everyone at ParadigmaXis.

I want to thank Professor Gabriel David for accepting the supervision of this project

I also want to thank my friends TC, Lin, Nitro, Vale, and my family.

Filipe de Lemos



*“ Man is distinguished, not only by his reason, but by this singular passion from other animals, which is a lust of the mind, that by a perseverance of delight in the continued and indefatigable generation of knowledge, exceeds the short vehemence of any carnal pleasure. ”*

Thomas Hobbes in *Leviathan*



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Project . . . . .	2
1.3	Objectives . . . . .	2
1.4	Overview . . . . .	2
<b>2</b>	<b>State of the Art</b>	<b>3</b>
2.1	OAIS Reference Model . . . . .	3
2.2	Repository Systems . . . . .	3
2.2.1	Fedora . . . . .	4
2.2.2	DSpace . . . . .	4
2.2.3	EPrints . . . . .	4
2.2.4	DigiTool . . . . .	5
2.3	Interoperability . . . . .	5
2.3.1	Definition of Digital Object . . . . .	5
2.3.2	Interoperability Protocols . . . . .	6
2.3.2.1	OAI-PMH – Protocol for Metadata Harvesting . . . . .	7
2.3.2.2	OAI-ORE – Object Reuse and Exchange . . . . .	7
2.3.2.3	SWORD – Simple Web-service Offering Repository Deposit . . . . .	8
2.4	Identification . . . . .	8
2.4.1	Properties of Identifiers . . . . .	8
2.4.1.1	Uniqueness . . . . .	9
2.4.1.2	Persistence . . . . .	9
2.4.1.3	Actionability . . . . .	9
2.4.1.4	Opacity . . . . .	10
2.4.2	Identification Systems . . . . .	10
2.4.2.1	URI – Uniform Resource Identifier . . . . .	10
2.4.2.2	URN – Uniform Resource Name . . . . .	10
2.4.2.3	HDL – Handle . . . . .	11
2.4.2.4	DOI – Digital Object Identifier . . . . .	11
2.4.2.5	PURL – Persistent Uniform Resource Locator . . . . .	11
2.4.2.6	ARK – Archival Resource Key . . . . .	12
2.5	Metadata . . . . .	12
2.5.1	Metadata Concepts . . . . .	12
2.5.1.1	Metamodel . . . . .	12
2.5.1.2	Metamodel Registry . . . . .	12
2.5.1.3	Metamodel Crosswalk . . . . .	13
2.5.2	Standard Metamodels . . . . .	13

## CONTENTS

2.5.2.1	ISAD(G) – General International Standard Archival Description	13
2.5.2.2	EAD – Encoded Archival Description . . . . .	13
2.5.2.3	METS – Metadata Encoding & Transmission Standard . . . . .	14
2.5.2.4	DC – Dublin Core . . . . .	14
2.5.2.5	PREMIS – Preservation Metadata: Implementation Strategies . . . . .	14
2.6	Search . . . . .	14
2.6.1	Search Approaches . . . . .	14
2.6.1.1	Centralized Search . . . . .	14
2.6.1.2	Distributed Search . . . . .	15
2.6.2	Search Standards . . . . .	15
2.6.2.1	SRW/SRU . . . . .	16
2.6.2.2	OpenURL . . . . .	16
2.7	Preservation . . . . .	16
2.7.1	Preservation Metadata . . . . .	17
2.7.2	Access Preservation . . . . .	17
2.7.2.1	Techno–Centric . . . . .	18
2.7.2.2	Process–Centric . . . . .	18
2.7.2.3	Data–Centric . . . . .	18
2.7.2.4	<i>Post hoc</i> . . . . .	18
<b>3</b>	<b>Comparative Study of Repository Systems</b>	<b>19</b>
3.1	Overview . . . . .	19
3.2	Interoperability . . . . .	20
3.2.1	API and Web Services . . . . .	20
3.2.2	Harvesting . . . . .	20
3.2.3	Search . . . . .	21
3.2.4	Ingestion . . . . .	21
3.2.5	Exportation . . . . .	21
3.2.6	External Authentication . . . . .	22
3.2.7	Metadata Standards . . . . .	22
3.3	Distribution . . . . .	22
3.3.1	Required Software . . . . .	22
3.3.2	Optional Software . . . . .	23
3.3.3	Installation and Packaging . . . . .	24
3.3.4	Updating . . . . .	24
3.4	Scalability . . . . .	24
3.4.1	Architecture . . . . .	24
3.4.2	Horizontal Partitioning – Sharding . . . . .	25
3.4.3	Horizontal Scaling – Replication . . . . .	25
3.5	Design . . . . .	25
3.5.1	Identification . . . . .	25
3.5.2	Addition of Digital Object Types . . . . .	26
3.5.3	Preservation . . . . .	26
3.6	Security . . . . .	27
3.6.1	Data Encryption . . . . .	27
3.6.2	Authentication . . . . .	27
3.6.3	Authorization . . . . .	27
3.7	UI Configuration . . . . .	28
3.7.1	Customization . . . . .	28

## CONTENTS

3.7.2	Internationalization and Localization . . . . .	28
3.7.3	Workflow integration . . . . .	28
3.7.4	Maintenance . . . . .	29
3.8	Summary . . . . .	29
<b>4</b>	<b>An Approach for Interoperability</b>	<b>31</b>
4.1	Interoperability Issues . . . . .	31
4.2	Interoperability Framework . . . . .	32
4.3	Interoperability Framework Approaches . . . . .	35
4.3.1	Identification of Digital Objects . . . . .	35
4.3.2	Harvest of Digital Objects . . . . .	36
4.3.3	Metadata Search . . . . .	37
4.3.4	Ingestion of Digital Objects . . . . .	37
4.3.5	Exportation of Datastreams . . . . .	37
4.3.6	Authentication . . . . .	38
4.4	Summary . . . . .	38
<b>5</b>	<b>Implementation of the Interoperability Framework</b>	<b>39</b>
5.1	Overview . . . . .	39
5.2	Abstract Classes and Interfaces . . . . .	40
5.2.1	AbstractRepository . . . . .	41
5.2.2	AbstractSet . . . . .	42
5.2.3	AbstractRecord . . . . .	42
5.2.4	InterfaceMetadatum . . . . .	43
5.2.5	InterfaceDatastream . . . . .	43
5.3	Base Classes . . . . .	44
5.3.1	Metadatum . . . . .	44
5.3.2	Datastream . . . . .	44
5.3.3	ArchiveSearch . . . . .	45
5.4	Adapters . . . . .	45
5.4.1	Repository Based on File URLs . . . . .	46
5.4.1.1	FileRepository . . . . .	46
5.4.1.2	FileRecord . . . . .	47
5.4.2	Repository Based on the OAI-PMH . . . . .	47
5.4.2.1	OAIRepository . . . . .	48
5.4.2.2	OAISet . . . . .	49
5.4.2.3	OAIRecord . . . . .	49
5.4.2.4	OAIDC . . . . .	50
5.4.3	Repository Based on the OAI-PMH and Fedora Access API Lite . . . . .	50
5.4.3.1	FedoraLiteOAIRepository . . . . .	51
5.4.3.2	FedoraLiteOAIRecord . . . . .	52
5.4.3.3	FedoraLiteOAIDatastream . . . . .	53
5.5	Applications . . . . .	53
5.5.1	OAIHarvest . . . . .	54
5.5.2	File Adapter GUI . . . . .	57
5.5.3	Repository Browser . . . . .	59
5.5.4	Repository Searcher . . . . .	60
5.6	Summary . . . . .	60

## CONTENTS

<b>6</b>	<b>Conclusions and Future Work</b>	<b>61</b>
6.1	Discussion . . . . .	61
6.2	Future Work . . . . .	62
	<b>References</b>	<b>68</b>
<b>A</b>	<b>Repository Systems Contacts</b>	<b>69</b>
A.1	Fedora . . . . .	69
A.2	DSpace . . . . .	69
A.3	EPrints . . . . .	69
A.4	Digitool . . . . .	69



# List of Figures

2.1	Mapping concepts from the OAIS Information Model to DSpace and Fedora systems, based on Figure 3 of [BVdS05]	5
2.2	Digital object.	6
4.1	Mapping of standard protocols and APIs by repository system and functionality.	32
4.2	Adapters as an interoperability layer between repositories and applications.	32
4.3	Relationships between the components in an adapter.	33
4.4	Relationship between records and digital objects.	33
4.5	Hierarchy of adapters.	34
4.6	Relationships between the components in an adapter.	35
4.7	Identification model of the framework.	35
5.1	Hierarchy of the implemented adapters.	39
5.2	Inheritance between the the classes of the implemented adapters.	40
5.3	AbstractRepository class.	41
5.4	AbstractSet class.	42
5.5	AbstractRecord class.	42
5.6	InterfaceMetadatum interface.	43
5.7	InterfaceDatastream interface.	44
5.8	Metadatum class.	44
5.9	Datastream class.	44
5.10	ArchiveSearch class.	45
5.11	Relationship between the classes in the File adapter.	46
5.12	FileRepository class.	47
5.13	FileRecord class.	47
5.14	Relationship between the classes in the OAI adapter.	48
5.15	OAIRepository class.	48
5.16	OAISet class.	49
5.17	OAIRecord class.	50
5.18	OAIIDC class.	50
5.19	Relationship between the classes in the OAI.FedoraLite adapter.	51
5.20	FedoraLiteOAIRepository class.	51
5.21	FedoraLiteOAIRecord class.	52
5.22	FedoraLiteOAIDatastream class.	53
5.23	Screenshot of the application OAIHarvest.	54
5.24	Activity diagram of the application OAIHarvest.	56
5.25	Activity diagram of the method HarvestRecords of the class OAIRepository.	57
5.26	Screenshots of the application File Adapter GUI.	58

## LIST OF FIGURES

5.27 Screenshot of the application Repository Browser. . . . .	59
--	----

# List of Tables

3.1 Summary of the Comparative Study of Repository Systems. . . . . 29

## LIST OF TABLES

# Abbreviations

AIP	Archival Information Package
API	Application Programming Interface
API-A	Fedora Access API
API-M	Fedora Management API
ARK	Archival Resource Key
CNRI	Corporation for National Research Initiatives
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management System
DC	Dublin Core
DIDL	Digital Item Declaration Language
DIP	Dissemination Information Package
DLL	Dynamic-Link Library
DNS	Domain Name System
DOI	Digital Object Identifier
DTD	Document Type Definition
EAD	Encoded Archival Description
ECL	Educational Community License
Fedora	Flexible Extensible Digital Object Repository Architecture
FEUP	Faculdade de Engenharia da Universidade do Porto
FOXML	Fedora Object XML
FTP	File Transfer Protocol
GISA	Gestão Integrada de Sistemas de Arquivo
GPL	GNU General Public License
GUI	Graphical User Interface
HDL	Handle
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol over Secure Socket Layer
ID	Identifier
ISAD(G)	General International Standard Archival Description
ISAAR(CPF)	International Standard Archival Authority Record for Corporate Bodies
ISBN	International Standard Book Number
ISO	International Organization for Standardization
JDK	Java Development Kit
JSP	JavaServer Pages

## LIST OF TABLES

LAMP	Linux Apache MySQL PHP/Perl/Python
LDAP	Lightweight Directory Access Protocol
MD5	Message-Digest algorithm 5
METS	Metadata Encoding & Transmission Standard
MIME	Multipurpose Internet Mail Extensions
NAAN	Name Assigning Authority Number
NMAH	Name Mapping Authority Hostport
OAI	Open Archives Initiative
OAI-ORE	Open Archives Initiative Object Reuse and Exchange
OAI-PMH	Open Archives Initiative Protocol for Metadata Harvesting
OAIS	Open Archival Information System
OCLC	Online Computer Library Center
OO	Object-Oriented
OOP	Object-Oriented Programming
OS	Operating System
PDF	Portable Document Format
PDI	Preservation Description Information
PID	Persistent IDentifier
PREMIS	Preservation Metadata: Implementation Strategies
PURL	Persistent Uniform Resource Locator
RDF	Resource Description Framework
Regex	Regular Expression
REST	Representational State Transfer
RFC	Request for Comments
SDK	Software Development Kit
SGML	Standard Generalized Markup Language
SIP	Submission Information Package
SQL	Structured Query Language
SRB	Storage Resource Broker
SSH	Secure Shell
SSL	Secure Sockets Layer
UI	User Interface
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
USMARC	United States MACHine-Readable Cataloging
UTF-8	8-bit UCS/Unicode Transformation Format
XACML	eXtensible Access Control Markup Language
XML	Extensible Markup Language

# Chapter 1

## Introduction

This chapter introduces the project discussed in this report. It starts by establishing its context and setting. Then it presents a short description of the project. It follows by defining a list of objectives to be satisfied. Finally it presents an overview of the document, showing the structure of this report.

### 1.1 Context

This project exists in the setting of the Gisa software application, developed by ParadigmaXis, and was done in the company's premises.

Gisa is a software application for the management and description of archives on several levels, ranging from the institution itself to specific parts of a document, following international archival descriptive standards – ISAD(G) and ISAAR(CPF). Clients of Gisa range from institutions with modest archives to institutions with large archives including the Archives Department of the Municipal Council of Porto and the Archive of Faculdade de Engenharia da Universidade do Porto (FEUP).

There is an increasing interest in digital archives for the preservation of digital documents. This includes institutions archiving documents that exist in digital form, and initiatives for the digitalization of documents.

Archives and their digital counterpart have similar reasons and functions. Yet now that the documents are digital, it is possible to couple these more closely with software applications. Therefore there is an interest in the integration between the software infrastructure components composing digital archives. This interest for integration is shared by the Gisa application, to establish a more prominent place as a system for digital archives.

The software infrastructure of a digital archive revolves around its repository, which stores the digital objects. Therefore it is mainly with the repositories that the focus of integration is turned to.

### 1.2 Project

This project is divided in two phases.

In the first phase, a comparative study is made to ascertain the characteristics of several open source repository systems for the deployment of repositories to serve as the base of digital archives. With this study it is possible to have a vision of the existing repository systems, their functionalities and methods to integrate with them. Additionally, it serves as a basis to suggest repository systems an institution can further study to create its digital archives.

In the second phase, a module is developed for the integration by Gisa with repositories of digital archives. This module should integrate with a repository system yet have enough flexibility to be extended for other repository systems.

### 1.3 Objectives

The objectives of this project are defined in the following list.

- Establish a list of parameters for the comparison between repository systems;
- Compare a selection of repository system, with particular emphasis on open-source systems;
- Develop an extensible module, to be used by Gisa, for its integration with a repository;
- Define an univocal identification scheme for the association with digital objects;
- Integrate in the module a search functionality on the contents of the digital archives;

### 1.4 Overview

This report is divided in the following six chapters. Chapter 1 introduces the project, its context and objectives. Chapter 2 presents the current perspectives, standards and technologies in the area of digital archives. Chapter 3 establishes a comparison between specific repository systems. Chapter 4 presents the issues in interoperability between repositories and introduces the architecture and approaches of the interoperability framework. Chapter 5 details the implementation of the interoperability framework and applications developed. Chapter 6 presents a summary of the project, its satisfaction of the objectives and directions for future work.



## Chapter 2

# State of the Art

Digital archives are archives dedicated to the preservation of electronic records. With the transition from the classical perspective of archives to the electronic realm, new challenges appear. This chapter presents the current perspectives, standards and technologies related to digital archives.

### 2.1 OAIS Reference Model

The OAIS Reference Model was developed by the Consultative Committee for Space Data System. This model is considered a landmark on the conceptualization of digital archives and defines general guidelines, as opposed to a prescriptive standard [Bal06].

An Open Archival Information System (OAIS) is an archive following the OAIS Reference Model. The term Open is used referring to the specification itself and not the access to the archives. Quoting the model itself [OAI02]:

“An OAIS is an archive, consisting of an organization of people and systems, that has accepted the responsibility to preserve information and make it available for a **Designated Community**.”

The OAIS Reference Model defines both an information and a functional model. The information model defines several entities around the concept of digital object and its composition (2.3.1). The OAIS Reference Model also gives particular attention to the preservation metadata of digital objects (2.7.1) and approaches to the preservation of access (2.7.2). This model also recognizes the need for interoperability between archives (2.3).

### 2.2 Repository Systems

Digital archives use repositories as the main software infrastructure. A repository is only a part of the ensemble of people and systems that form digital archives.

Repository systems are software frameworks to build repositories for digital archives.

The repository systems Fedora, DSpace and EPrints were selected for this work on the basis of being Open-Source, being considered proven software and still in active development. DigiTool was also included to introduce a proprietary repository system, which is in use at the Library of FEUP.

Chapter 5 further describes and compares each of the selected repository systems.

[Pow05] describes the services a repository should provide and listing of possible protocols and standard to implement these services in an interoperable manner. The most important aspects of these services are presented in the following sections. These are interoperability (2.3); identification (2.4); metadata (2.5); search (2.6); and preservation (2.7). For each aspect a selection of standards are also presented.

### 2.2.1 Fedora

Fedora is a repository system developed at Cornell University since 1997. It has no specific audience nor built-in frontend [Fed]. The homepage introduces Fedora with:

“Fedora open source software, a robust integrated repository-centered platform that enables the storage, access and management of virtually any kind of digital content.”

### 2.2.2 DSpace

DSpace is a repository system, resulting of the HP-MIT Alliance. It was first released in 2002. It is aimed for institutional repositories [DSp]. The homepage introduces DSpace with:

“DSpace captures your data in any format – in text, video, audio, and data. It distributes it over the web. It indexes your work, so users can search and retrieve your items. It preserves your digital work over the long term. DSpace provides a way to manage your research materials and publications in a professionally maintained repository to give them greater visibility and accessibility over time.”

### 2.2.3 EPrints

EPrints is a repository system aimed for institutional repositories and scientific journals [EPr]. The origin of EPrints begins in 2000, with the Digital Repositories programme of Joint Information Systems Committee (JISC). The homepage introduces EPrints with:

“EPrints is the most flexible platform for building high quality, high value repositories, recognized as the easiest and fastest way to set up repositories of research literature, scientific data, student theses, project reports, multimedia artefacts, teaching materials, scholarly collections, digitised records, exhibitions and performances.”

## 2.2.4 DigiTool

DigiTool is developed by Ex Libris as a repository system to manage digital resources. DigiTool is aimed for libraries and can manage disparate collections of digital resources, such as images, thesis, e-research data, learning and course materials [Dig]. The homepage introduces DigiTool with:

“DigiTool® enables academic libraries and library consortia to manage and provide access to digital resources, both those that are created for use within the institution and those that are collected and maintained by the library for the benefit of the public.”

## 2.3 Interoperability

The OAIS model expects the need for interoperability between archives [OAI02, 6 Archive Interoperability].

The proposed scale to identify the level of interoperability, in ascendant order, is the following: **Independent; Cooperating; Federated; and Shared resources.**

The first issue related with the interoperability between digital archives is the very definition of digital object.

### 2.3.1 Definition of Digital Object

The definition of the concept of digital object and related terms is question to some debate and may vary depending on the context.

One possible definition is that a **digital object** is a uniquely identified unit composed of data and metadata.

Figure 2.1 shows the mapping of concepts by [BVdS05] of digital object from the OAIS Information Model to DSpace and Fedora. Mappings to the aDORe repository were also provided, but this repository is not studied in this work [VdSBL<sup>+</sup>05].

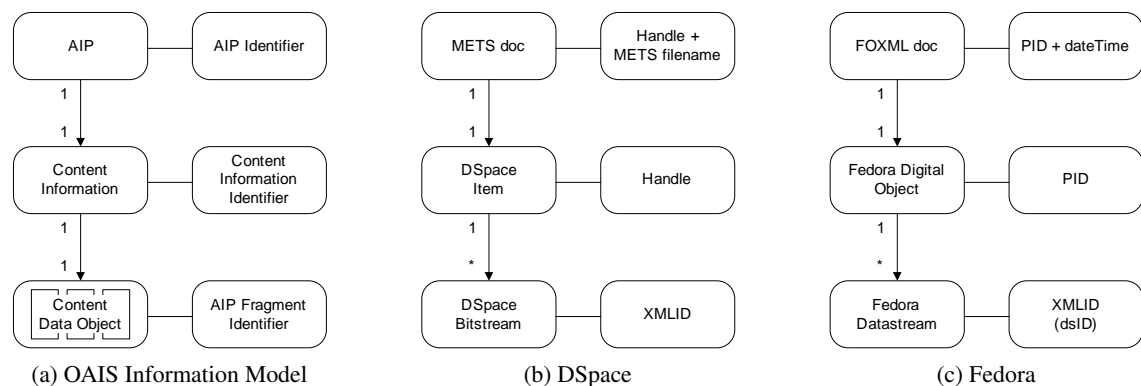


Figure 2.1: Mapping concepts from the OAIS Information Model to DSpace and Fedora systems, based on Figure 3 of [BVdS05]

The terms used in this work take as reference the ones used by the Fedora repository system, and are shown in Figure 2.2.

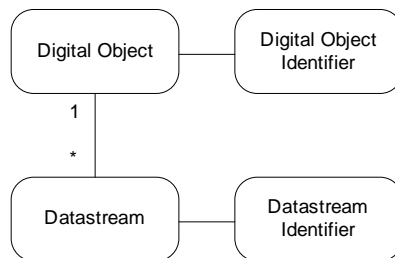


Figure 2.2: Digital object.

The term **datastream** is used for an accessible chunk of data of a digital object. Datastreams are usually files in a file system.

The term **metadata** is used for any data about the digital object and its datastreams. No assumptions about the content of metadata are made at this point.

These are generic terms used to transmit the intended message without introducing several different terms for the same concept. Taking as an example the very definition of digital object used by the California Digital Library [Col06]:

“Digital Object: An entity in which one or more **content files** and their corresponding metadata are united, physically and/or logically, through the use of a digital wrapper.”

Instead of using the new term “content files”, which would only add a layer of complexity and obfuscate further concepts, the equivalent term “datastreams” can be used instead. When in a specific context, the original term is kept and appropriate equivalent terms referred.

According to the OAIS Reference Model, there are three types of **Information Packages** distinguished by the process they exist in: Archival; Submission; and Dissemination.

An **Archival Information Package** (AIP) is a normalized structure with Content Information (digital object) and the associated Preservation Definition Information (PDI).

A **Submission Information Package** (SIP) is sent by a producer, to be placed in the repository as an Archival Information Package (AIP). The transformation process is required as a SIP might be incomplete and need additional SIPs or have insufficient PDI and need additional processing.

A **Dissemination Information Package** (DIP) is provided as a request from a consumer. It might be the entirety or part of an AIP or a transformation of it.

### 2.3.2 Interoperability Protocols

For interoperability between digital archives to exist, the protocols used must be defined. Using standard protocols facilitates compatibility and interoperability between applications [Pow05].

### 2.3.2.1 OAI-PMH – Protocol for Metadata Harvesting

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a protocol to allow the collection of metadata descriptions from digital archives resources [LdSNW02b]. The appearance and generalization of OAI-PMH has made the creation of archive federations easier.

There are two main actors: the Provider, which exposes the metadata, and the Harvester, which collects the metadata. These are the server and client respectively of a client-server architecture. The Harvester issues the requests to the Provider using a REST-like interface and the Provider returns the appropriate reply in a XML document.

The harvest is made around six requests(or verbs): Identify; ListMetadataFormats; ListSets; ListRecords; ListIdentifiers; and GetRecord.

**Identify** — Used to retrieve information about the repository.

**ListMetadataFormats** — Used to retrieve the metadata formats available in the repository. Unqualified Dublin Core is mandatory but other metadata formats are encouraged [LdSNW02a].

**ListRecords** — Used to harvest records from a repository. A record contains the identifier and metadata of a resource, in the format specified in the request. Selective harvesting can be made by specifying a set the records must belong to, and timestamps, retrieving only records modified from/until then.

**ListSets** — Used to retrieve the list of record collections, reflecting the structural organization of the resources in the repository.

**ListIdentifiers** — Similar to ListRecords, but only the identifiers are retrieved.

**GetRecord** — Used to retrieve a single record from the repository. The parameters include the identifier of the item, and metadata prefix.

The OAI-PMH is being used in a multitude of projects involving interoperability, such as Arc [LMZN01], MOSC [Roe05] and OAIster [Hag03] to name a few. It is used for unconventional repositories, such as a thesaurus, a digital library usage logs, and an OpenURL registry[VdSYH03].

However, the OAI-PMH is defined for the harvest of digital object metadata but not their datastreams. This functionality is covered by a protocol currently being developed – the Open Archives Initiative Object Reuse and Exchange (OAI-ORE).

### 2.3.2.2 OAI-ORE – Object Reuse and Exchange

Open Archives Initiative Object Reuse and Exchange (OAI-ORE) is a protocol in development to describe and exchange aggregations of Web resources. As of June 2008, a final specification is still in development. A beta version of the OAI-ORE specification was released in June 2008. The development of libraries using the beta specification has been announced [For08].

The protocol is based around the ORE Model which uses Resource Maps to aggregate resources that are identified as URIs. These aggregations can be used to map the set of datastreams

of a digital object in a repository. This is even explicitly stated by the initial specifications, in the context of digital libraries [[LdSJ<sup>+</sup>08a](#)]. An example of an aggregation is given as:

A scholarly publication stored in an ePrint repository such as arXiv or in a DSpace, ePrints, or Fedora repository. Such a publication may appear on the Web as multiple resources, each with an individual URI. The set of resources typically consists of a human readable "splash page", that links to the body of the publication in multiple formats such as LaTeX, PDF, and HTML.

Resource Maps can be represented in several forms: Atom Syndication Format; RDF/XML; and RDFa. The distribution of Resource Map can be done using: OAI-PMH; XML SiteMaps; and Atom Syndication Feeds [[LdSJ<sup>+</sup>08b](#)].

The combination of the OAI-PMH and OAI-ORE protocols, should allow a more precise harvesting of the content of a repository – obtaining the record information down to the datastreams of the digital objects.

### 2.3.2.3 SWORD – Simple Web-service Offering Repository Deposit

SWORD is a protocol that allows the deposit, in a repository, of digital objects and collections of digital objects [[AFL08](#)]. It is built on top of the Atom Publishing Protocol (APP), focusing on the deposit of files and extends APP to support additional parameters.

Implementations of SWORD were made for several repository systems, including Fedora, DSpace and EPrints [[AFL08](#), Technical Outputs and Proof of Concept]. For these repositories it supports the deposit of simple files and METS documents with several files. The created digital object will contain the metadata provided and these files as datastreams.

## 2.4 Identification

An identifier is a string of characters that names a resource. Identification is an important aspect of digital archives, as it is the identifier of a digital object that allows references to it. There are several approaches to identification, more or less suitable depending on the concept, intention and technology.

### 2.4.1 Properties of Identifiers

Identifiers can be characterized on the dimensions of: **uniqueness**; **persistence**; **actionability**; and **opacity**. Identifiers to be used for long term identification of digital objects should be unique, persistent, actionable and opaque [[Coy06](#)].

### 2.4.1.1 Uniqueness

Uniqueness restricts that, within a context, an identifier refers to one and only one resource [Coy06]. A **globally unique** identifier is unique across systems, opposed to an identifier that is unique just inside one repository.

When expanding the use of identifiers that are not globally unique, outside of their initial context, there can be collisions between identifiers. For example, two different repositories might have the same identifier for unrelated resources. This issue can be avoided by having the identifier of an object associated with the identifier of the repository to make a compound identifier for the digital object that is globally unique.

### 2.4.1.2 Persistence

Persistence refers to the longevity of the identification – that is, the association of the identifier with the resource. The persistence of this association is the responsibility of the organization managing the resource. Therefore it depends of a service provided by the organization rather than the identifier technology itself [Kun03, SWJF96]. The resource that a persistent identifier points to should remain the same during its lifetime. As such a persistent identifier must not be reused.

Persistence also implies that the identifier is independent of the location of the resource, because otherwise, if the location changes so will the identifier. For example, generic URLs are not persistent identifiers because if the location of a file changed, so will the URL. In this case a Persistent URL (PURL) would remain the same – only the resolution service would reflect the change with the redirection destination. The difference between a generic URL and a PURL is the organizational commitment by the OCLC to preserve the the PURL and its association to the resource, even if the location of it changes [Coy06]. PURLs are further explained in Section 2.4.2.5.

### 2.4.1.3 Actionability

Actionability refers to the ability of resolving the location of the identified resource. An identifier is actionable when it is possible to obtain the resource through a service.

The actionability dimension regards the binding of the identifier to the resolution service [pro07], which can be:

- fixed to a resolution protocol, even if the it might not be active to a particular identifier instance (PURLs; and ARKs);
- implicit resolution protocol, that is bound to an explicit service at presentation time and may change according to the context (Handles; DOIs; URIs in the case of URLs that are usually, but not necessarily, bound to HTTP);
- explicit resolution protocol, that allows the choice of the service and may even provide a list of possible resolution services for an identifier (OpenURLs; URNs like urn:isbn and urn:ietf:rft, that define a resource like a book and an RFC, but not the location; alternative implementations of Handles).

#### 2.4.1.4 Opacity

Opacity refers to metadata content in the identifier itself [Wei06]. There are cases for which this is desirable, as in domain names of URLs – for example `http://www.google.com/` is easier to remember than `http://66.249.93.104/`. However, in identifiers with low opacity, changes in this metadata either break the identifier or make it inconsistent. This issue is a generalization of one previously mentioned – having the location present in the identifier itself.

For instance, the popular arXiv.org e-Print archive changed its identifier scheme in April 2007 to address this [arX07]. The format `<archive>.<subject class>/<year><month><number>` used until then, resulted in identifiers like `arXiv:math.GT/0309136`. However, this had a low opacity which made changes in the classification difficult. To address this issue, the identifier format was changed to `<year><month><number>` giving identifiers like `arXiv:0706.0001`. This has a higher opacity and is more flexible. Yet this format is still not completely opaque, as it keeps the year and date, which might bring up similar issues.

### 2.4.2 Identification Systems

The concept of identifiers has been implemented using several methodologies in different systems. Some identification systems include: **URIs**; **URNs**; **Handles**; **DOIs**; **PURLs**; **ARKs**.

#### 2.4.2.1 URI – Uniform Resource Identifier

An URI is a sequence of characters, using a defined syntax, that identify a resource. For example an URL like `http://www.google.com/webhp?hl=en` is an URI used for the localization of a resource. The latest specification is defined in [BLFM05].

The URI identifier syntax is the following:

```
<scheme name>:<hierarchical part>[?<query>] [#<fragment>]
```

The `<scheme name>` adds specifications to the format of the identifier.

URIs are unique, not inherently persistent nor actionable, and have a variable level of opacity.

#### 2.4.2.2 URN – Uniform Resource Name

An URN is an URI using the scheme name “urn:”. It identifies resources in a persistent and independent way [Moa97]. The introduction of namespaces that share these characteristics is easy. For example, an ISBN in this form would be `urn:isbn:0060012366`. The URN specification builds on the URI. Their functionalities and use are different, however.

The URN identifier syntax is the following:

```
urn:<namespace identifier>:<namespace specific string>
```

The format of `<namespace specific string>` depends on the namespace.

URNs are independent from the resource location. For example, the RFC about the syntax of URN can be identified by the URN `urn:ietf:rft:2141` while there are several URLs for the resource, one being `http://www.rfc-ref.org/RFC-TEXTS/2141/`.



URNs are unique, persistent, not actionable and have a variable level of opacity.

#### 2.4.2.3 HDL – Handle

The Handle System, developed by the Corporation for National Research Initiatives (CNRI), is an infrastructure for the creation of identification systems. These identification systems defined by a prefix and use a server for the generation and resolution of identifiers with that prefix [[HDL](#)].

The HDL identifier uses the syntax:

```
[hdl:/]<prefix>/<resource code>
```

The prefix and resource code identify the resource with global uniqueness. The identifiers can be written as URIs using the scheme name “hdl:”. The specifications for the Handle System are defined in the RFCs 3650, 3651 and 3652.

Example of an identifier, [hdl:4263537/4000](#), that resolves to [The Handle System](#).

HDLs are unique, persistent, actionable and opaque.

#### 2.4.2.4 DOI – Digital Object Identifier

The DOI System is a deployment of the CNRI Handle System with the prefix “10.”, followed by a specific code of a DOI entity attributed by the International DOI Foundation. DOIs can be resolved to the resource they identify by a DOI System Proxy Server which communicates with the Handle System [[DOI](#)].

The DOI identifier uses the syntax:

```
[doi:/]10.<DOI entity code>/<resource code>
```

DOIs in the form of URIs use the scheme name “doi:”.

Example of an identifier, [doi:10.1000/182](#), that resolves to [The DOI® Handbook](#), a document further explaining the DOI System.

DOIs are HDLs, and DOI uses the Handle System, so their main characteristics are the same – DOIs are unique, persistent, actionable and opaque.

#### 2.4.2.5 PURL – Persistent Uniform Resource Locator

PURLs are URLs pointing to a redirection service. This service uses HTTP redirects to the actual URL of the resource [[SWJF96](#), [PUR](#)]. While this has the advantage of being inherently actionable, it is dependent on technologies like HTTP and DNS, which compromises long term preservation.

The PURL identifier uses the syntax:

```
http://<resolver address>/<name>
```

Example of an identifier, [http://purl.oclc.org/NET/eisa/40](#), that resolves to [The prime numbers](#) in the Encyclopedia of Integer Sequences.

PURLs are unique, persistent, actionable and have a low opacity.

#### 2.4.2.6 ARK – Archival Resource Key

System of identifiers that takes into consideration the subjacent technological aspects of actionability [Kun03].

An ARK identifier uses the syntax:

[http://NMAH/]ark:/NAAN/Name[Qualifier]

In the above syntax, the protocol and NMAH (Name Mapping Authority Hostport) are optional and can change. If they exist, then the ARK is encapsulated in the form of an URL. This way it can be resolved to the resource, using a browser for example. This part makes the identifier actionable. The components “ark:”, NAAN (Name Assigning Authority Number) correspond to the code of the generating entity. This part ensures the identifier is globally unique. The code of the generating entity, together with the Name, identify the object.

ARKs are unique, persistent, actionable and opaque.

## 2.5 Metadata

Metadata is information about the data. Taking as example a digital photograph – the data are the bytes composing the image, while the metadata include the name of the author; the date the photograph was taken; the exposure time; the focal length used, and one could go on being as descriptive as wished. Storekeeping the metadata of a digital object is a fundamental part of digital archives, as it is with the metadata that the characteristics of a digital object can be known.

### 2.5.1 Metadata Concepts

There are three concepts related to metadata, which are important in digital archives: metamodel, metamodel registry and metamodel crosswalk.

#### 2.5.1.1 Metamodel

Metamodels are typically a set or hierarchy of key-value fields specifying the format of the metadata. There is a large amount of models to store metadata, each with its own objectives and fields. These fields can be classified in categories according to their purpose [ISO03]: **identification; definition; relational; representational; administrative.**

#### 2.5.1.2 Metamodel Registry

A metamodel registry is a centralized location where metamodels are stored and maintained. ISO/IEC 11179 provides the international standard for the representation of a metadata definition in a Metadata Registry [ISO03].

Metamodel registries are related with long-term preservation, as it is by the understanding of metamodels that these retain their value.

### **2.5.1.3 Metamodel Crosswalk**

Interoperability between metamodels can be defined as the reuse of overlapping conceptual spaces, and when fields apply to the same concept they are interchangeable. Crosswalks define the process to transform metadata from one model to another. Problems may arise when dealing with different metamodels. Depending on the compatibility, the use of a crosswalk can incur in loss of information. This can occur if fields are different in descriptiveness, or if there are complex fields in the source model that map to more than one field in the target model, or if fields do not even have an equivalent in the target model.

For example, a repository might operate only on Dublin Core (DC) metadata. If it was to ingest a digital object containing EAD metadata, a mapping to DC would be required. However EAD has a much greater expressive power than DC so loss of information is bound to happen. A possible solution would be to crosswalk where possible and keep the original EAD metadata in a adequately identified datastream of the digital object. The original metadata might be later used or crosswalked as needed.

[BP01] demonstrated the development of a distributed metadata registry framework that allows to maintain several metamodels and define crosswalks between them.

## **2.5.2 Standard Metamodels**

There is a multitude of metamodels, therefore this section only cites a few that are used in this work.

### **2.5.2.1 ISAD(G) – General International Standard Archival Description**

Standard for the archival of documents defining the elements composing the descriptive information [ISA99].

The metamodel used in the Gisa application follows the ISAD(G) specification.

The first edition of ISAD(G) was released in 1996 and reviewed in 2000.

### **2.5.2.2 EAD – Encoded Archival Description**

Standard in XML to describe searchable archives [EAD]. The specification also defines crosswalks to map metadata elements of EAD with ISAD(G) and, less accurately, with Dublin Core and USMARC.

The high level of interoperability between EAD and ISAD(G), makes it a good candidate for the serialization of metadata in XML in the Gisa application.

The first version was released in 1998 as a SGML DTD. The second version was released in 2002 as a XML DTD.

### 2.5.2.3 METS – Metadata Encoding & Transmission Standard

Standard in XML for structural metadata of digital objects that allows the association with files and other types of metadata, like PREMIS for example [MET].

### 2.5.2.4 DC – Dublin Core

Standard that defines the elements for the description of resources [DC]. The base set contains 15 elements, approved as ISO15836, NISOZ3985, RFC 5013. Qualified Dublin Core expands the base set with new elements, refinements and encoding schemas. The specification is independent from an implementations, but it is typically used with XML or RDF.

Unqualified Dublin Core is a very simple and general metamodel, as such it can be used in many different contexts, at the cost of expressive power.

Dublin Core results from a workshop in 1995.

### 2.5.2.5 PREMIS – Preservation Metadata: Implementation Strategies

Specification of a Data Dictionary with a model and elements for preservation metadata [PRE]. There is an XML Schema for the codification of the PREMIS Data Dictionary.

The Final Report of the first version was released on May 2005. Version 2 was released on March 2008.

## 2.6 Search

In the digital archiving context, the objective of a search is to retrieve, from one or more repositories, the list of relevant records based on a query on their metadata.

### 2.6.1 Search Approaches

Searching through a set of repositories can take two approaches: **centralized search**; and **distributed search**.

#### 2.6.1.1 Centralized Search

The centralized search approach requires the use of browse services to crawl through the repositories to harvest the required metadata of the documents. The harvested metadata is stored locally and indexed so that the search service may answer queries on that metadata and deliver the relevant results [Inf02].

Large search engines, like Google, use distributed crawling, where a large quantity of computers realize the harvesting and indexing of webpages, to provide a centralized search service.

The disadvantages of centralized search when compared to distributed search are:

- it requires the existence of a harvesting/browsing service (like OAI-PMH);

- the metadata might incur loss of information during the harvesting;
- the metadata might become outdated;
- it needs to perform harvests to check for updates;
- it requires additional resources like disk space and memory;
- difficulties might arise in scaling to store the data of many large repositories.

#### **2.6.1.2 Distributed Search**

The distributed search (also known as federated search or metasearch) approach decentralizes the actual search through the several repositories. Each of them provides an interface to their searching service. A distributed search calls every interface with that query, eventually normalizing the results and removing/merging duplicates [Inf02].

To reduce the complexity of the problem, it would be favorable if these interfaces are based on standard protocols, such as Z39.50 or OpenSearch, and in the ideal case use the same protocol. However it is typically assumed that these search interfaces are heterogeneous. If so the initial query has to be translated for each interface with a variable degree of complexity and fidelity.

Solutions have been developed to face the implementation issues of centralized searches in digital libraries [LC06].

The disadvantages of distributed search when compared to centralized search are:

- it requires the existence of available search services;
- the dependence on other systems as a subset of repositories might be slow/overloaded and delay the results;
- the possible loss of information in the results from normalization;
- special features present in some search services and interfaces might not be usable if they're not present in every repository searched through;
- the high complexity of constructing personalized queries, normalizing the results and finding duplicates;
- establishing the federation of repositories might also be a more difficult task than creating a harvester.

#### **2.6.2 Search Standards**

The distributed search approach is greatly simplified if the target repositories implement standard search protocols.

### 2.6.2.1 SRW/SRU

SRW/SRU stands for a search service with two interfaces: SRW, Search and Retrieve Web Service, which binds with SOAP interface; and SRU, Search and Retrieve via URL, which binds with a REST-like interface [SRU]. Both interfaces expect XML documents as results.

It uses Common Query Language (CQL), a standard language for the representation of the queries. The semantics are based on the Z39.50 protocol.

While Z39.50 uses its own communication protocol anterior to the internet, SRW/SRU uses HTTP for both SOAP and REST-like interfaces.

### 2.6.2.2 OpenURL

OpenURL differs from the general concept of search standard, yet can be seen as one as it is used to retrieve a resource based on its description [Ope]. Alternatively it can be seen as an exportation protocol, if the digital object is known to exist and an unique identifier is used.

The OpenURL system works by making a query to the OpenURL Framework Application in order to obtain a context-sensitive resource. The query package contains: a descriptor of the resource, such as an identifier and title, the context where the resource is referenced and the context of the service. This way OpenURL can be used to return the most appropriate copy of a resource.

## 2.7 Preservation

The long term preservation of digital information is one of the main concerns of a digital archive. Preservation is a complex problem that includes several related aspects.

There are two environments for preservation: *in vitro*, controlled and institutional environment; and *in vivo*, occurring in the "living web" outside a digital archive [KMSN07, HN06]. While the main focus of Digital Archives is on *in vitro* preservation, some of the following works for the *in vivo* preservation of web sites are worth mentioning for the methodologies presented.

[MSN06, MN06] proposed a "lazy preservation" methodology and developed a web crawler that uses search engine caches. It can ultimately be used for the reconstruction of entire web sites in the case of hardware failure. Additionally they developed metrics for the evaluation of the fidelity of the preserved copy and evaluated different policies for crawling taking in account issues of request efficiency and missing resources.

[HN06] presented a "just-in-time" methodology with a similar purpose, to cope with 404 responses from web servers. By using lexical signatures of pages it can search for similar versions of a page.

[SN08] proposed a methodology where the web server itself is responsible for the creation of objects ready to be stored for preservation. This technique is supported by a module in the web server that exposes the digital object using OAI-PMH. At the time of dissemination utilities generate metadata that is sent with the resource. Particular attention is given to the technical metadata derived from file format recognition.

### 2.7.1 Preservation Metadata

The OAIS model defines the metadata necessary for the preservation of a digital object as Preservation Description Information (PDI) [OAI02, 2.2.2 Information Package Definition]. The categories of PDI are: **Provenance**; **Context**; **Reference**; and **Fixity**.

These categories can be defined as:

- Provenance – administrative metadata; Stores the source or origin of the information, the history of changes and custody.
- Context – relational and some definition metadata; Stores the relationships of the object with its environment: the reason why it was created and relationship with other objects. The definition metadata which doesn't belong to the Context is covered in the OAIS model by the Content Information.
- Reference – identification metadata; Stores internal identifiers for the content of the object and external, unambiguous identifiers for the object.
- Fixity – representational metadata. Supports data integrity mechanisms such as file checksums (for example MD5).

The preservation of the digital objects implies not only storing the data and metadata but also the ability to recognize the format and access the information. The ability to recognize the format of information is provided through adequate **technical metadata**. Technical metadata are a subset of preservation metadata that can be classified as representational.

From the need of recognition arises the importance given to the Representation Information in the OAIS model [OAI02, 2.2.1 Information Definition]. It is through this information, and the Designed Community Knowledge Base, that the Data Object (datastream) can be opened and viewed. For example, to view a datastream the only information required could be the associated MIME type string “text/plain”. However this might not be enough to confidently preserve the digital object. Additional Representation Information might be needed such as the character encoding (for example ASCII, ISO-8859-1 or UTF-8) and the type of newline used: LF (used by Unix-like OS); CR+LF (Windows); CR (Mac OS).

Metamodels aimed for preservation metadata include PREMIS (2.5.2.5).

### 2.7.2 Access Preservation

Even if the format of the content stored in the datastream is recognized, one still needs to be able to access the information. This requires hardware and software that is able to use that format. However, as file formats become outdated, they are replaced and application support for them is prone to disappear. There are several approaches to preservation [Thi02], which can be aggregated in four basic groups [Wil08]: **techno-centric**; **process-centric**; **data-centric**; and *post hoc*.

Each approach has both support and opposition, especially data-centric migration *versus* process-centric emulation [Gra00].

### 2.7.2.1 Techno–Centric

The techno-centric approach focuses on keeping the original hardware and software as long as possible. This approach can be found in digital museums, for example. This access preservation approach keeps the setting as is therefore it limits integration with newer technologies.

### 2.7.2.2 Process–Centric

The process-centric approach focuses on keeping the original applications and processes.

The methods used for access preservation in this approach are based on **porting** the application to the new environment or **emulation** of the hardware and software. The OAIS model considers this approach and differentiates its stance on it depending on the availability of the original application source code. If source code is available then porting it is a viable possibility, to be confirmed through adequate testing. Alternatively, bridges might exist to refresh the datastreams to a new version of the application – this is a version migration, a data–centric approach, favored over emulation. In the case where neither source code nor commercial bridges are available, emulation or another approach must be used [OAI02, 5.2.2.1 Methodologies Involving Source Code Availability]. About emulation, the OAIS specification details some aspects of this approach and points out the limitations and difficulties in the creation of such emulators [OAI02, 5.2.2.2 Potential Emulation Approaches].

### 2.7.2.3 Data–Centric

The data-centric approach focuses on keeping the information, at the expense of the original application.

The main process used is **migration** – a Transformation Migration in OAIS parlance, as there are changes to the Content Information or PDI. This can range from version migration to format standardization – converting the content of digital objects stored in obsolete or closed file formats to open contemporary formats. A classic example is the conversion of text documents to PDF, a widely supported open format which guarantees the reading functionalities at the expense of edition [Gra00]. This example brings to light the aspect of significant properties – differentiating the functionalities that need preservation (reading the document in the previous example) from those dispensable (writing). Significant properties are content and context specific, and must be carefully asserted before migration, under the risk of losing valuable information [Wil08].

### 2.7.2.4 *Post hoc*

*Post hoc* (after the fact) is the preservation approach by default. Nothing is done regarding preservation and, when needed, is the data analyzed and extracted using “data archaeology” techniques.

*Post hoc* is the cheapest approach on the short term, but most expensive on the long term when a late migration is done and old, possibly incompatible and unknown hardware and software must be used.



## Chapter 3

# Comparative Study of Repository Systems

Comparisons between repository systems have been made in several studies, such as in [W<sup>+</sup>06] and [FCR<sup>+</sup>07, DSpace VS Fedora]. These are of great value, yet there is no definitive comparison due to the constant development of the software and the fact that studies rate repositories based on their own specific requirements.

This study gives a special focus to the interoperability aspects (API, web services, protocols implemented) and general characteristics of each repository system.

The results of this and prior studies can be taken into account for a first analysis in the selection of a repository system, to be used in the development of a Digital Archive. A deeper study of the requirements for the repository and features offered by each repository system must still be made by each individual institution, to ascertain the most suited one.

The choice of a repository system, to build a repository, can be influenced by the product license, either open-source or commercial. The type of open-source license is also important – there are differences in restrictions between a strong copyleft free software license like GPL and a permissive free software license like BSD. GPL establishes that derived works are licensed under a compatible copyleft license [GPL].

The repository systems studied were Fedora (2.2.1), DSpace (2.2.2) and EPrints (2.2.3). All these are Open Source. Additionally, DigiTool (2.2.4), a commercial software, was included in the Interoperability section (3.2).

Fedora is exclusively a repository system and does not include a frontend interface. Therefore it was excluded from UI Configuration section (3.7), as it deals mostly with aspects of the frontend. Third-party frontends have been developed for Fedora, such as Fez [Wea05] and ELATED [ea05].

### 3.1 Overview

**Fedora** — It has an open source (ECL) license. Version 2.2.1 was used in the study. Documentation and reports on the Version 3.0 Beta 1 and 2 were also taken in consideration.

Uses JAVA in the backend and does not include frontend.

**DSpace** — It has an open source (BSD) license. Version 1.5.0b1 was used in the study. Uses JAVA in the backend and JSP in the frontend.

**EPrints** — It has an open source (GPL) license. Version 3.0.3 was used in the study. It is completely built in Perl.

**DigiTool** — It is proprietary software with a commercial license. Documentation of version 3 and an instance of a repository using DigiTool were used in the study.

## 3.2 Interoperability

This section studies the existence of APIs and standard protocols in each repository.

### 3.2.1 API and Web Services

**Fedora** — The Fedora API is made of web services that can be accessed using both REST and SOAP interfaces over the network [[Com08](#), Fedora APIs]. The API is also divided by functionalities: Access and Management.

**DSpace** — Since it is open source, it is possible to access the API directly using Java. It is divided in Content Management, Search, Harvesting and Browse [[DSp08](#), Architecture]. There is also LNI, Lightweight Network Interface, with less functionalities but easier to deploy over a network [[Sto06](#)].

**EPrints** — Since it is open source, it is possible to access the API directly using Perl. It is divided by subjects: Core, DataObjects, Session, Search + List, Misc, Metadata, Plugins [[EPr07](#), API]. As of version 3, web services are under development but not yet functional. [[EPr08](#), Web Services]

**DigiTool** — Web services with SOAP access or a REST-like interface. The SOAP API is divided in three parts: Digital Entity Manager; Digital Entity Explorer; Metadata Manager [[ExL08](#)]. X-Server can be accessed using a REST-like interface [[ExL06](#)].

### 3.2.2 Harvesting

**Fedora** — OAI-PMH, assured by the OAI Provider Service [[Fed07e](#)].

**DSpace** — OAI-PMH and Harvesting API (which the OAI-PMH uses) [[DSp08](#)].

**EPrints** — OAI-PMH [[EPr07](#), OAI].

**DigiTool** — OAI-PMH.

### 3.2.3 Search

The OpenURL standard can be considered either a search or an exportation protocol. In this work it is listed under search, but the exact nature of it can be argued.

**Fedora** — Generic Search Service (GSearch) is a Fedora service that creates and searches an index of FOXML records using as backend either Lucene; Solr; or Zebra [Fed07d]. Also includes an OpenURL interface.

**DSpace** — Search API wrapping on the Lucene search engine. Supports OpenURL built on top of the search system [DSp08, Search].

**EPrints** — Search using the web UI and OpenURL.

**DigiTool** — There is the Digital Entity Search API in the Digital Entity Explorer web service [ExL08]. X-Server also includes query functionalities [ExL06]. Supports OpenURL.

### 3.2.4 Ingestion

There are implementations of the SWORD Version 1.0 protocol for Fedora, DSpace and EPrints.

**Fedora** — Ingestion can be made by using the API, Fedora Administrator or via command line [Fed07b, Ingest and Export of Digital Objects]. There is also the Directory Ingest Service – a service to ingest a hierarchical directory of files into the repository [Fed07c].

**DSpace** — Ingestion can be made by either a Web Submit UI or a Batch Item Importer which uses Package Ingestor Plugins to extend the supported package types [DSp08, DDK07].

**EPrints** — Ingestion can be made using the web UI; using web services (for example: PubMed or CrossRef); or using the appropriate Import plugin on a SIP.

**DigiTool** — Ingestion can be made by using the web submit UI or using the create command in the Digital Entity Call API of Digital Entity Manager [ExL08].

### 3.2.5 Exportation

**Fedora** — Exportation, like ingestion, can be made by using the API, Fedora Administrator or via command line [Fed07b, Ingest and Export of Digital Objects].

**DSpace** — Exportation of objects can be made using the web UI, API, via command line or with a tool that export items as METS metadata and referenced datastream files [DSp08, Exporting Items].

**EPrints** — Export plugins can disseminate just metadata or the entire DIP [EP07, XML Export Format]. The exportation of datastreams individually seem to require the development of a plugin. The exportation can also be made from a web UI page.

**DigiTool** — Web UI, X-Server [ExL06, Viewing Objects] or Digital Entity Manager [ExL08].

### 3.2.6 External Authentication

**Fedora** — Supports LDAP [[Fed07b](#), Security Architecture - Servlet Filters for Authentication].

**DSpace** — Supports LDAP by placing the corresponding plugin in the authentication stack [[DSp08](#), Configuring LDAP Authentication]. Other plugins can be used [[DSp08](#), Custom Authentication].

**EPrints** — Supports LDAP [[EPr08](#), Integrating EPrints with LDAP] and Central Authentication Service (CAS) [[EPr08](#), CAS].

**DigiTool** — Supports single sign-on with external authentication servers including LDAP [[ExL05](#)].

### 3.2.7 Metadata Standards

Repository systems can store metadata of any format as files, which are accessible as datastreams. This section deals with support of specific metamodels as such.

**Fedora** — Supports Unqualified Dublin Core, mandatory as a datastream for every digital object [[Fed07b](#), Overview: The Fedora Digital Object Model]. Supports ingestion of SIPs in METS and FOXML [[Fed07b](#), Ingest and Export of Digital Objects].

**DSpace** — Supports Unqualified Dublin Core and METS. Additional Metadata representations used can be ingested and disseminated with Crosswalk Plugins [[DSp08](#)]. Existing crosswalks include MODS, Qualified Dublin Core, DIDL.

**EPrints** — Supports Dublin Core, METS, MODS, DIDL. Import and export plugins for other formats exist and additional plugins can be developed.

**DigiTool** — Supports several metamodels, including Dublin Core, MARC 21 and METS [[ExL06](#), [ExL08](#)].

## 3.3 Distribution

This section studies the technical requirements for the deployment of each repository system. All repository systems studied have moderate hardware requirements – a server machine and sufficient hard disk drive space for the content.

### 3.3.1 Required Software

**Fedora** — Fedora distributions include most of the required software [[Fed07a](#), 1.1. Install Prerequisites], except:

- Java SE Development Kit (JDK) 5.0

**DSpace** — The required software is portable for all OS DSpace runs on. [[DSp08](#), Installation]

- Java 1.4 SDK
- Apache Ant 1.6.x
- PostgreSQL 7.3 or Oracle 9
- Apache Tomcat 4.x/5.x or Jetty or Caucho Resin or equivalent server

**EPrints** — The required software [[EPr08](#), Required Software] are:

- Apache HTTP Server 2.2.4
- MySQL 5.0
- ActivePerl 820 on Windows
- Perl, mod\_perl, GDOME on Unix-like OS

### 3.3.2 Optional Software

**Fedora** — It is possible to use alternatives to the provided database and server. [[Fed07a](#), 1.1. Install Prerequisites]

- MySQL or Oracle 9 or PostgreSQL — Database alternative to the embedded McKoi SQL Database 1.0.3.
- Jetty or JBoss — Servlet Container alternative to the embedded Tomcat 5.0.28. This can be any servlet container that implements Servlet 2.4/JSP 2.0 or higher.
- Ant 1.6.5 or later — Build environment used to compile from source.

**DSpace** — The activation of some features require the following software. [[DSp08](#), Installation]

- Perl — Allows the generation of statistical reports.
- CNRI Handle Server — Allows the internal use of CNRI handles identifiers. It is included in the DSpace distributions but the activation is optional.

**EPrints** — Several optional functionalities of EPrints require additional software. On Windows these are:

- GhostScript 8.60 — Allows the indexing and previewing of PostScript and PDF files.
- Catdoc 0.94.2 — Allows the indexing and previewing of PostScript and PDF files.
- ImageMagick 6.3.5-6 — Allows conversion and indexing of image formats.

### 3.3.3 Installation and Packaging

**Fedora** — Can be installed on Unix-like OS or Windows, since Fedora is a Java application. An installer '.jar' file is provided [Fed07a, 1.2 Download Fedora]. Compiling it from source is a viable option too. [Fed07a, 6. Installation from Source]

**DSpace** — Can be installed on Unix-like OS or Windows. Needs compilation of source code using Apache Ant. Installation instructions and dependencies are explained in the documentation [DSp08, Installation].

**EPrints** — Can be installed on Unix-like OS, OS-X or Windows with prepared packages, instructions and dependency lists [EPr08, Installing Eprints 3 on Windows].

### 3.3.4 Updating

**Fedora** — The minor version updates vary depending on source and target versions. There are instructions and some tools for these updates [Fed07b, Fedora Upgrade and Migration Guide]. There are instructions for the major version migration of Fedora 1 to Fedora 2 [Fed05, Fedora Migration Utility Guide].

**DSpace** — Updating DSpace is no trivial operation and the degree of complexity varies with the source and target versions. Prior updating instructions are carefully explained in the documentation [DSp08, Updating a DSpace Installation].

**EPrints** — Minor version updates are usually done by the installer and there are instructions to save custom content. For major version updates there are conversion scripts: from 1 to 2; from 2 to 3 [EPr08, Migration].

## 3.4 Scalability

This section deals with the ability of a repository system to deal with large amounts of data.

### 3.4.1 Architecture

**Fedora** — Being a web service, the architecture of Fedora allows distribution from start. Performance-wise there is an expected overhead due to the layers of SOAP messages, which require parsing of XML documents, and services. Since the prototype version, Fedora is expected to scale well up to 10 million objects [Fed07b, Introduction to Fedora]. A recent initiative to test the performance and scalability of Fedora showed that it can, for example, ingest objects with as much as 10,000 inline datastreams [eSc08].

**DSpace** — DSpace has had reports of scalability issues for version 1.4 related to the architecture itself [W<sup>+</sup>06]. Release 1.5 improves on this aspect especially in the browsing system [DSp08, Changes in DSpace 1.5].

**EPrints** — Web application based on the LAMP (Perl) architecture. It is simple and separates the web application and database.

### 3.4.2 Horizontal Partitioning – Sharding

**Fedora** — Fedora supports distributed content objects. This is done by defining the type of content of a datastream, either as External Referenced Content or Redirect Referenced Content [Fed07b, Overview: The Fedora Digital Object Model]. Additionally SRB-based Low Level Storage allows the integration with SRB for the storage of contents, with all its advantages on distribution [RZ06].

**DSpace** — Alternatively to storing objects on the file system, DSpace easily integrates with SRB [Dsp08, Storage Resource Broker (SRB) Support].

**EPrints** — Datastreams are stored as files and most EPrints web pages are statically generated. This static content can be cached in a **proxy** box using applications such as Squid. However sharding the **database** would require modifications to the model.

### 3.4.3 Horizontal Scaling – Replication

**Fedora** — Compared to version 2.2.1, version 3.0 Beta 2 gives a greater attention to the replication of objects [Fed08, The Fedora Digital Object Model]. Fedora Data Objects can be replicated between repositories. Using the same identifier simplifies the process through which a resolver system can distribute the load amongst the participating repositories.

**DSpace** — No special replication features for DSpace were found.

**EPrints** — The MySQL **database** includes replication functionalities - it is possible to add slave servers handling read requests [MyS]. These should include the great majority of operations on the repository.

## 3.5 Design

This section deals with specific design characteristics of the repository systems.

### 3.5.1 Identification

**Fedora** — Uses PIDs, persistent identifiers unique in the repository, in the namespace:object form [Fed07b, Fedora Identifiers]. They can be written as an URI by prepending “info:fedora/”. Around these, a URI can be built to identify a datastream or dissemination by appending datastream IDs and method calls. The PID Generator extension allows the use of PIDs based on the Handle system.

**DSpace** — Typically uses the CNRI Handle System, for which a repository instance of DSpace should obtain a prefix from the CNRI [DSp08, Handles]. It can be configured to use other identifiers such as PURLs. Identifiers are assigned to sets (communities and collections) and digital objects (items). For the persistent identification of datastreams (bitstreams and bundles) of a digital object there is a unique Sequence ID for each datastream in an object.

**EPrints** — Uses a numerical ID code, uniquely identifying a digital object within a repository.

### 3.5.2 Addition of Digital Object Types

**Fedora** — Fedora has strong functionalities around datastream handling by use of disseminators. The datastream type can be identified with by its MIME type. Associating disseminators to the object, it is possible to completely define the dissemination, therefore adding a new type of digital object and its behavior. [Fed07b, Overview: The Fedora Digital Object Model]

**DSpace** — Adding recognized datastream formats is done in the Bitstream Format Registry that stores for each format a name, description, MIME type, file extensions, support level (unknown, known, supported) and a administrative purpose flag to hide format from user [DSp08, Bitstream Format Registry].

**EPrints** — Directly out-of-the-box it supports a large spectrum of digital object types with previewing and adequate dissemination of documents (PDF, Microsoft Word, Excel) and multimedia (images, audio, video). Metadata fields vary with the object type. Additional types require manual customization of the database, but there are plans for the creation of a tool for this task.

### 3.5.3 Preservation

**Fedora** — Starting from version 2.2 the repository can compute, store and check checksums of datastreams [Fed07b, Checksums on Datastreams in Fedora]. This can be used to verify the integrity of the digital object. The automatic computation of checksums can be enabled to run at every datastream change. The general behavior can be overridden for particular cases. The Fedora Rebuild utility can, in case of repository corruption, rebuild it by crawling the digital objects XML source files stored on disk [Fed07b, Repository Rebuilder Utility].

**DSpace** — Checksums of all datastreams are stored at ingestion. DSpace includes a tool, Checksum Checker, that verifies the integrity of the repository by doing a total or partial match of the repository files against their checksum [DSp08, Checksum Checker].

**EPrints** — As of version 3, EPrints comprises the following preservation mechanisms: updates the 'preservation metadata' with the changes on the repository object (History Module); allows preservation service to download of all files and metadata of an object (METS or DIDL); notifies preservation service of licenses [EPr07, Preservation Support].



## 3.6 Security

This section deals with the aspects of security, user authentication and authorization of repository systems.

Security aspects of a repository might be viewed as unimportant in the context of digital archives, since its use is mostly internal. This study leaves it up to the institutions to decide whether or not to consider the security aspects of the repository system used.

### 3.6.1 Data Encryption

**Fedora** — The application servlet (Tomcat by default) can be configured during installation to enable SSL [[Fed07a](#), 2.2. Custom Install].

**DSpace** — The application server can be configured to enable HTTPS [[DSp08](#), DSpace over HTTPS].

**EPrints** — HTTPS can be used through adequate configuration of Apache and its module `mod_ssl`. This is well documented, although the instructions for HTTPS on EPrints 3 also report some bugs exist [[EPr08](#), Secure EPrints3 with HTTPS].

### 3.6.2 Authentication

**Fedora** — Modular and extensible authentication mechanism, using servlet security filters. [[Fed07b](#), Security Architecture - Servlet Filters for Authentication]

**DSpace** — Modular and extensible authentication mechanism, called Stackable Authentication: there is a stack of authentication methods successively called until one succeeds, yet all of them can still recognize the user as belonging to groups [[DSp08](#), Authentication].

**EPrints** — Modular and extensible authentication mechanism, allows the edition of user types (defaults being 'user', 'editor', 'admin') and setting different authentication methods and permissions for these types. Authentication can be made using the basic access authentication of HTTP with username and password [[EPr07](#), Login-Only Repository].

### 3.6.3 Authorization

**Fedora** — Supports XACML and enforces its policies [[Fed07b](#), Fedora Authorization with XACML Policy Enforcement]. Authorizations can be fine-grained to the level of APIs; objects; datastreams; and disseminators.

**DSpace** — High granularity of authorization [[DSp08](#), Authorization]. Permission attributes are associated with groups of users, with two special groups: 'administrators' with every permission and 'anonymous' encompassing all users. ADD/REMOVE permissions can be defined at the levels of Community, Collection, Item and Bundle, while READ permissions can be defined down to the Item and Bitstream level. However, Metadata is always visible.

**EPrints** — Authorization is done by user type, to which a set of permissions is granted. These have a moderate level of granularity and include permissions for user management; 'deposit', allowing the submission of items in the archive; 'editor', allowing to edit or delete records. [EPr07, Repository Configuration, 4.2 User Privs]

## 3.7 UI Configuration

This section deals with the ability to configure the user interface for each repository system. As previously mentioned, Fedora is only a backend and, as such, is left out of this section.

### 3.7.1 Customization

**DSpace** — There are two UI: One based on JSP where simpler changes can be made by editing the configuration files. More complex changes can be made to the Web UI by editing the JSP files themselves [DSp08, JSPUI Interface Customization]. The alternate UI is based on the Apache Cocoon framework and is called Manakin. It allows flexible customization of themes [DSp08, XMLUI Interface Customizations (Manakin)].

**EPrints** — Allows customization, ranging from simple changes like using the Institution logo, colors or theme, by editing the configuration file and templates, to redesigning the user interface, which requires changing XHTML and CSS [EPr07, Branding]. To put these changes in effect, the related pages need to be rebuilt using appropriate commands.

### 3.7.2 Internationalization and Localization

**DSpace** — Internally uses UTF-8 encoding. Uses Java Resource Bundles to have the text in locale-specific files. There are language packs with localizations to several other languages.

**EPrints** — Internally uses UTF-8 encoding. EPrints allows multiple languages on an installation and has a flexible internationalization system having all phrases in XML files. It is translated in several languages [EPr07, I18n].

### 3.7.3 Workflow integration

**DSpace** — Simple workflow, defined at the collection level, composed of up to 3 steps. In a submission, at each step the associated group is notified and can either reject it and return it to the submitter, or accept it and pass it on to the next step, until the end, where it is accepted in the main archive. If a step has no group associated then the process skips to the next step. [DSp08, Workflow System]

**EPrints** — Prior to version 3, only a minimalist workflow could be used where, after a paper was submitted, it was moderated by the repository staff when it could have been rejected and returned to the submitter, or accepted and placed in a publicly accessible location. Since

EPrints 3, it is possible to define a more complex workflow, with a custom input form and set of stages than can be optional and object type specific [EPr07, Workflow Format].

### 3.7.4 Maintenance

**DSpace** — There are features that require scripts to be regularly run. These include alerts to e-mail subscribers for new content, the generation of thumbnails and extraction of the text from documents for indexing, vacuuming of the PostgreSQL database, generation of statistical reports, integrity verification. These can be automated using cron jobs/scheduled tasks [DSp08, 'cron' Jobs]. Windows systems can use the similar Scheduled Tasks.

**EPrints** — Frontend webpages and abstracts are not automatically updated with the repository, needing to apply commands to regenerate these [EPr07, Generate Scripts]. These can be automated using cron jobs, or Scheduled Tasks on Windows [EPr07, Automating your maintenance].

## 3.8 Summary

Table 3.1 summarizes the comparative study. To each characteristic is given a rating for the perceived satisfaction of the general requirements for a repository. The rating can be either +, ++ or +++, for a poor, an average or a good rating, respectively. A blank space means the repository system was not studied for that characteristic.

Table 3.1: Summary of the Comparative Study of Repository Systems.

	Fedora	DSpace	EPrints	DigiTool
(3.1) License	ECL	BSD	GPL	commercial
(3.2.1) API and Web Services	+++	++	+	+++
(3.2.2) Harvesting	+++	+++	+++	+++
(3.2.3) Search	+++	+++	++	+++
(3.2.4) Ingestion	+++	+++	+++	+++
(3.2.5) Exportation	+++	+++	++	+++
(3.2.6) External Authentication	+++	+++	+++	+++
(3.2.7) Metadata Standards	+++	+++	+++	+++
(3.3) Distribution	+++	+++	+++	
(3.4) Scalability	+++	++	+++	
(3.5) Design	+++	+++	++	
(3.6) Security	+++	+++	+++	
(3.7) UI Configuration		+++	+++	

Fedora has the best backend, but no frontend for the end user. DSpace is also a good repository system. EPrints scored slightly lower due to the lack of some interoperability features. These two systems include a frontend especially geared towards digital libraries. DigiTool has strong features for the interoperability with other repositories or tools.



## Chapter 4

# An Approach for Interoperability

The interoperability of applications with repositories raises some issues. These issues are more visible when it is necessary to operate over several different repositories.

This chapter presents a framework intended to provide a systematic, reusable method for the development of interoperability layers between repositories and applications.

### 4.1 Interoperability Issues

Interoperability between repositories is hindered by the following issues:

- Divergences in the data model of digital objects. Even when based on the OAIS model, the mapping between the concept of digital object between repositories is different.
- Differences in the identification of digital objects across repositories. Uniqueness between repositories cannot be presumed. Persistence can also be questioned, especially in identifiers with low opacity.
- Difficulties in obtaining the list of digital objects in a repository and their metadata. This is related with the data model and metadata model used.
- Complexity of searching across the metadata of digital objects on several repositories.
- Discrepancies in the access of the datastreams in digital objects.
- Coordination of different authentication methods and parameters.

To address these issues there are standard protocols implemented by repositories. However their use requires a closer examination. By mapping standard protocols and APIs studied in the previous chapter, it is possible to visualize their coverage, as presented in Figure [4.1](#).

## An Approach for Interoperability

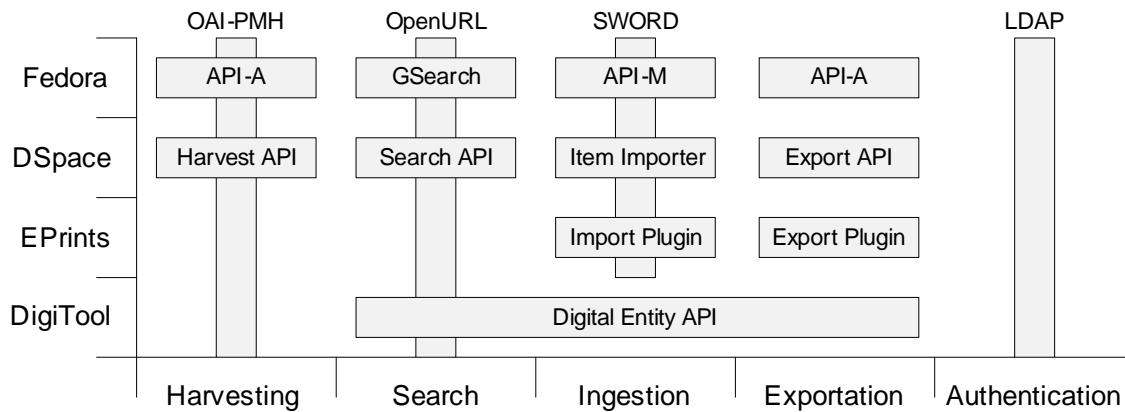


Figure 4.1: Mapping of standard protocols and APIs by repository system and functionality.

In the vertical dimension are the repository systems studied: Fedora; DSpace; EPrints; and DigiTool. In the horizontal dimension are the interoperability functionalities of repositories: Harvesting; Search; Ingestion; Exportation; and Authentication. APIs of repositories systems (horizontal bands) only work with a specific type of repository. Standard protocols (vertical bands) are designed for a single functionality. Interoperability can require functionalities spanning through the scope of several standard protocols.

It is visible that, even for similar functionalities, protocols diverge in their existence and implementation across repositories.

With these issues in mind, a framework for interoperability was developed.

## 4.2 Interoperability Framework

This work introduces the architecture of an Interoperability Framework. This framework was developed taking into account the issues explained in the previous section and can be used to achieve a certain degree of interoperability between an application and repositories of digital archives. The framework, concepts and models it uses are explained in this section.

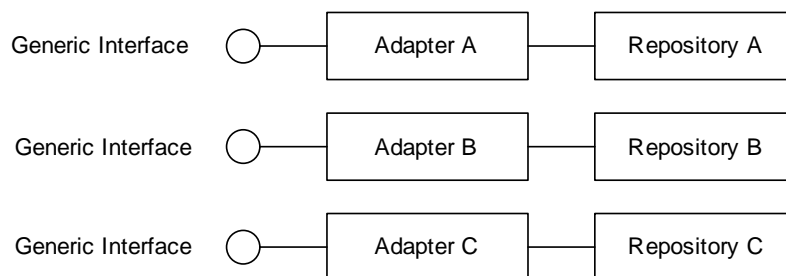


Figure 4.2: Adapters as an introperability layer between repositories and applications.

An **adapter** is a layer that knows the repository and its protocols. Adapters encapsulate the business logic specific to operate the repository it represents, such as its location on the network, the protocols used to access its contents and the data formats it uses. This way an adapter would cover an entire horizontal band in figure 4.1.

The adapter also presents a more abstract, generic interface. This generic interface is common to all adapters. It can be used to work on several repositories at the same time and using the same application.

While differing internally, at the generic interface level of abstraction, all adapters share the same data model. This data model is based on the components expected of a repository and its digital objects. The components are: **repository**; **set**; **record**; **metadatum**; and **datastream**. The relationships between them are presented in Figure 4.3.

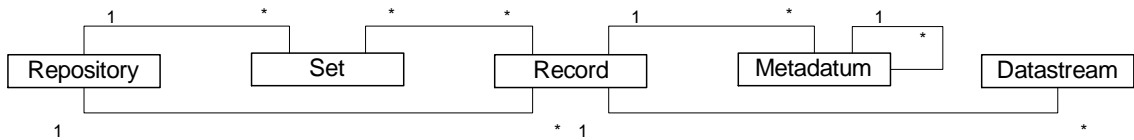


Figure 4.3: Relationships between the components in an adapter.

**Repository** — It is a stand-in, in the adapter side, for the repository as a whole. This component contains the records, both in a flat structure and optionally in a set structure.

**Set** — It is a collection of record components, reflecting the organization of the digital objects in the repository. A record can be in none, one or many sets. Sets can contain references to other sets to form a hierarchical structure. A set does not implicitly contains the records of its subsets.

**Record** — A record component is a view of a digital object of the repository. This relationship between a record and a digital object can be seen in Figure 4.4.

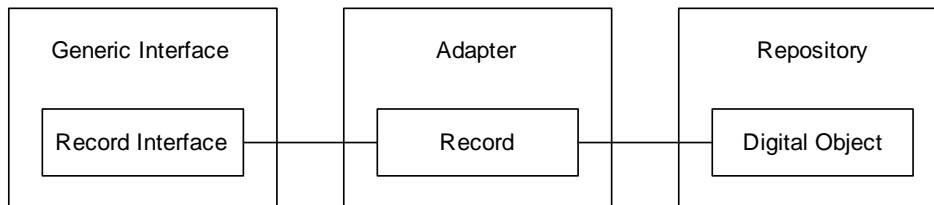


Figure 4.4: Relationship between records and digital objects.

The name “record” was used instead of “digital object” to highlight that it is only a representation of it. The original repository maintains the control and responsibility over the original digital object. A record can be a transformation in relation to the original digital

object – for example use a different metadata format – and in this aspect it is closer to the DIP concept in the OAIS model (2.3.1).

**Metadatum** — A metadatum component is an unit of metadata of the record. It is an individual entry of metadata in the key-value form. At the generic interface level, there are no restrictions placed on either value. Records are expected to have several metadatum instances, even though, technically, they might have none. Additionally metadatum can have references to other metadatum instances in the same record, to form a hierarchical structure.

**Datastream** — A datastream component represents one datastream of the digital object. As such these components are expected to be able to access the datastream in the original repository. At the generic interface level, datastream components are considered independent from each other.

Adapters encapsulate the used protocols to achieve the required functionalities – both standard protocols and repository API. Adapters present their generic functionalities in the same manner, independently of the specific protocol and implementation.

The advantage of standard protocols is that implementations exist for many different repositories. Examples of these are the protocols SWORD and OAI-PMH, as seen in the repository system comparison (3.2).

Repositories using the same repository system also have common services. For example all instances of Fedora repositories expose the same Fedora APIs.

Therefore, instead of using the linear relationship presented in Figure 4.2, adapters can also be seen in a hierarchy. By using a hierarchical relationship between adapters it is possible to display common functionalities through an additional interface, as seen in Figure 4.5.

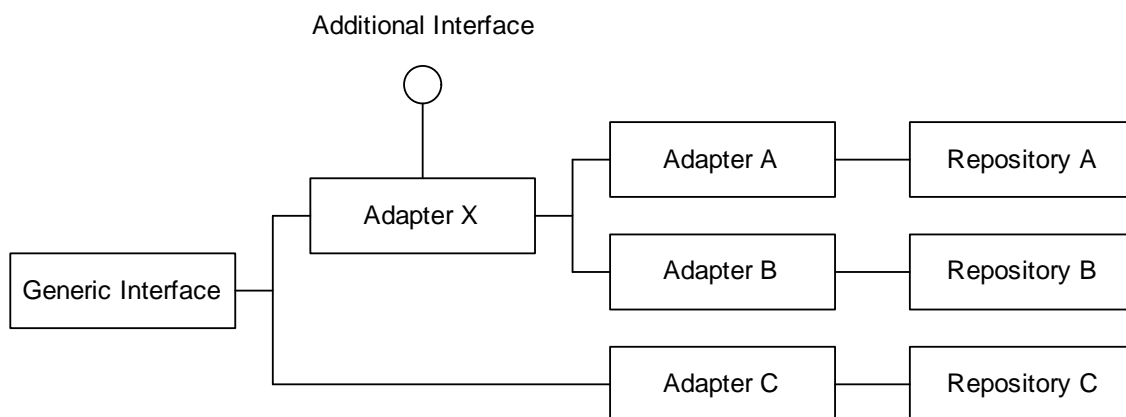


Figure 4.5: Hierarchy of adapters.

The additional interface can introduce functionalities at a lower level of abstraction compared to the generic interface. These functionalities might be required for some applications but are hidden from those only using the generic interface. This view on the structure of adapters also



allows a much greater flexibility in the architecture and should reduce the effort required to create new adapters.

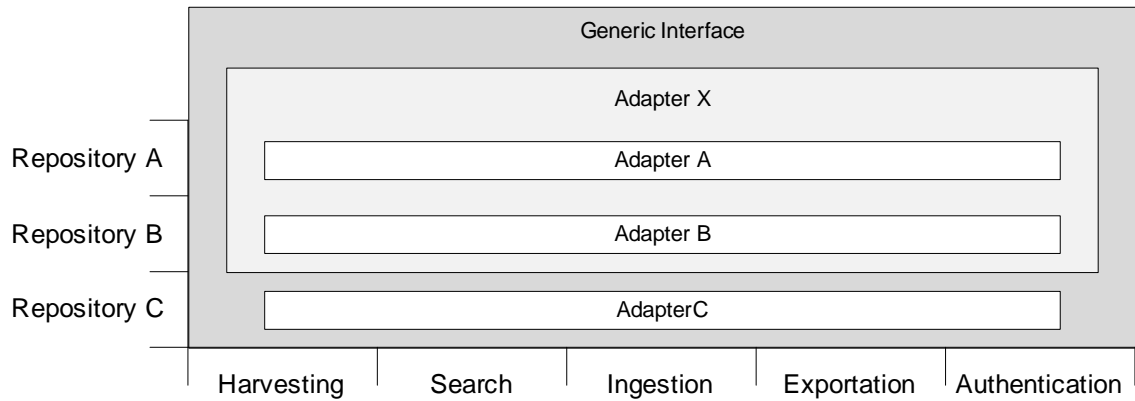


Figure 4.6: Relationships between the components in an adapter.

Figure 4.6 presents the intended coverage of adapters in a diagram similar to Figure 4.1. The generic interface itself can be seen as the most abstract adapter, exposing the common functionalities of all adapters.

### 4.3 Interoperability Framework Approaches

This section bases itself on the interoperability framework and data model introduced in the previous section, and presents additional approaches to answer some of the issues raised in 4.1.

#### 4.3.1 Identification of Digital Objects

The homogenous identification of digital objects is needed for the association between digital objects and documents in the Gisa application.

The identification model is presented in Figure 4.7, by using a fictitious arXiv adapter connected to the arXiv repository.

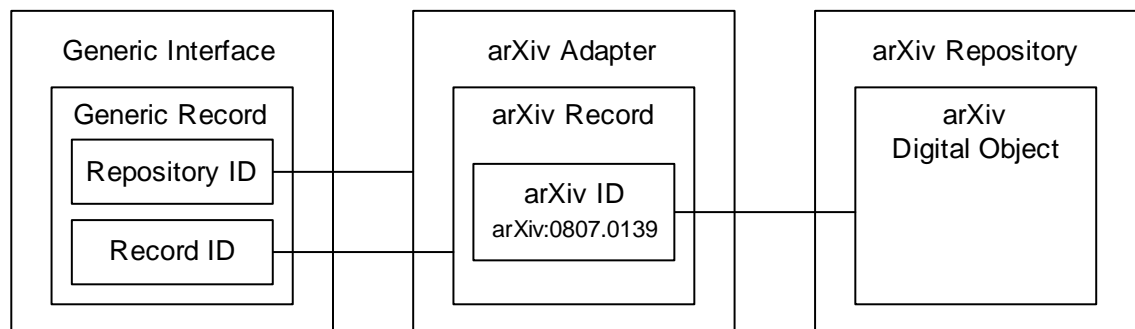


Figure 4.7: Identification model of the framework.

In this example, the arXiv identifier (arXiv:0807.0139) is stored and used internally by the arXiv record component to associate itself with the original digital object. Since identifiers vary between repositories this identifier is not used for the generic interface. Instead a compound identifier is used.

The compound identifier Repository ID + Record ID uniquely identifies the record. These two identifiers are integers and are present in all repository and record components for all adapters. Repository ID is the identifier of the adapter instance corresponding to the repository of the record, and is unique amongst all other adapters. It is needed to ascertain the identity of the correct adapter to obtain the record from. Record ID is the identifier of the record in the adapter, and must be unique amongst all other records in that adapter.

In this approach, by adding a new indirection it is possible to identify records in the framework in an homogenous manner. Responsibility for storekeeping the original identifier is passed to the implementation of the record component. Should this identifier be required at a later date, it should be obtainable at the level of abstraction of the specific adapter.

### 4.3.2 Harvest of Digital Objects

One of the initial issues is how to obtain the list of available digital objects in a repository.

This list can be obtained by two approaches. The difference between them is when the list is obtained – it can be obtained either in the "present", by using a protocol at runtime, or in the "past", by using stored results of a previous harvest. An adapter can use either of these approaches.

The first approach requires the existence of protocols that cover the necessary browsing and searching functionalities, including: returning lists of digital objects with unique and constant identifiers; filtering these lists by sets; counting the available objects in the repository; obtaining the corresponding metadata; and search functionalities in this metadata.

The main advantage of this approach is the freshness of the list. It is always up to date. However few repositories expose the necessary protocols to external applications. Even if protocols exist, coordinating these operations is expected to be more complex than using the alternative approach.

The second approach is to make an harvest of the available digital objects, and obtain their identifiers along with their metadata. The results are stored, to be used at a later date. When browsing and searching functionalities are needed, they are done on the harvested data. This approach is used in all implemented adapters to this date. It has the disadvantage of requiring the harvest prior to its usage. Additionally, the data can always be considered out-of-date. To minimize this, incremental harvests to update the data can be made at regular intervals and by request.

For the harvest approach the most appropriate standard protocol is OAI-PMH. As seen previously, it is implemented in many repositories.

### 4.3.3 Metadata Search

Search functionalities are an important feature in repositories. These allow to find specific objects in a large ensemble. A query is made and the objects with matching metadata are returned.

A search in this framework can be run in one or more repositories simultaneously. It can be seen as both a centralized and a distributed search, depending on the level of abstraction.

It can be seen as a centralized search because it uses the data of several repositories, which are centralized and homogenous when seen from the generic interface perspective.

At a different level of abstraction it can also be seen as a distributed search, because the search runs in each adapter instance. The matching records are returned in an homogenous manner at the generic level, in the form of a list of records.

This difference in levels of abstraction is reflected in the search functionalities offered by the framework.

A simple free-text search is run over all the metadata of the objects in the involved repositories.

A finer grained control over the fields searched is also offered according to the keys of key-value metadata. The metadata keys vary between repositories and metamodels. Therefore their use in searches is made at the adapter level of abstraction. For example a search on metadata with the key "author" returns no records in repositories where no such metadata key exists. Repositories with this metadata key return records with a matching metadata value for this key.

At the generic level there is no knowledge about which fields exist. At lower abstraction levels, specific metadata keys or even different search methods can be specified in additional interfaces.

### 4.3.4 Ingestion of Digital Objects

So far the interoperability framework has been explained for read-only functionalities. Creating and editing are also functionalities which might be included in this framework. This type of interaction is, at the moment, not covered by the generic interface, nor has it been implemented.

At a lower level of abstraction it might be useful to create or edit new digital objects in repositories. The SWORD protocol can be used to create new digital objects in a repository. Changes made in the records of an adapter can be reflected in the repository by using the API of the repository.

### 4.3.5 Exportation of Datastreams

The exportation of the actual datastreams of a digital object has two moments. The first one is obtaining the list of available datastreams of an object. The second one is obtaining the datastream itself.

Currently there is no standard protocol for this functionality, but the finalization of the OAI-ORE protocol and its implementation in repositories might change this in the near future.

The best approach is to use the specific API of a repository to obtain the list of datastreams. Each datastream usually has an URL, or an identifier which can be used to construct an URL, to obtain its location. By using the corresponding URL, it is possible to obtain a datastream over

the defined protocol. This can be any protocol supported like HTTP, HTTPS, FTP or even the file system.

Alternatively some repositories return the datastreams in a single file (DIP). This file, can be cached and processed to list the datastreams it contains and allow the access to them.

OpenURL can also be considered for the exportation of digital objects. However, it provides little control over the datastreams.

### **4.3.6 Authentication**

The responsibility of authentication is handed to the adapter. When an adapter uses protocols with the repository it represents, it might require authentication. The adapter itself contains the business logic to do this authentication. At the generic interface level, it is seen as if no authentication is required.

For example, a username and password might be required to access the datastreams of a repository. These authentication parameters are stored in the adapter itself. When needed, the adapter provides these values in the appropriate manner.

This way the framework can cope with different authentication methods and values.

## **4.4 Summary**

This chapter presented the common issues of interoperability in repositories: divergences in the data model; identification; harvesting; search; ingestion; exportation; and authentication.

An interoperability framework was presented that addresses these issues by using adapters to repositories over different levels of abstraction.

Finally, several approaches to answer the issues raised were presented using this framework.

## Chapter 5

# Implementation of the Interoperability Framework

This chapter describes the implementation of the interoperability framework introduced.

### 5.1 Overview

The interoperability framework is implemented as a White Box Framework, where additional functionalities are added by deriving existing classes.

The programming language used is C#, which is also used by Gisa. The object-oriented programming (OOP) paradigm of C# suits the framework architecture, using concepts of class, inheritance, abstraction and polymorphism. The classes and interfaces are encapsulated in a C# DLL, also known as C# Class Library.

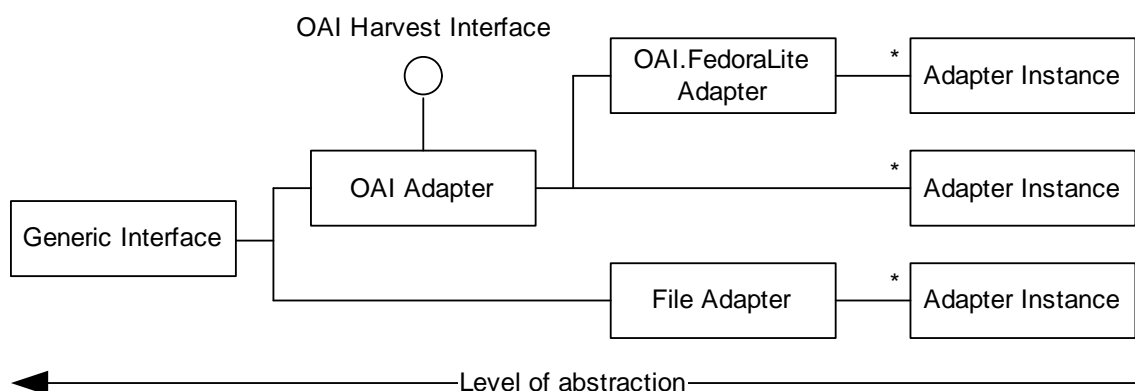


Figure 5.1: Hierarchy of the implemented adapters.

## Implementation of the Interoperability Framework

Each component is a class – repository, set, record, metadatum, datastream. Applications can see them through the same generic interface, while the actual implementation is done in adapters.

In this implementation, an adapter is a set of classes prepared for a specific type of repository or protocol. Each class in an adapter derives by inheritance or implements the interface from the generic interface. The generic interface can be seen as the most abstract adapter, from which all the others are derived.

Three adapters are implemented: File adapter; OAI adapter; and OAI.FedoraLite adapter. Any of these three adapters can be instantiated to adapter instances for specific repositories, as seen in Figure 5.1.

The inheritance relationships between the classes can be seen in Figure 5.2. Each adapter is contained in a separate namespace.

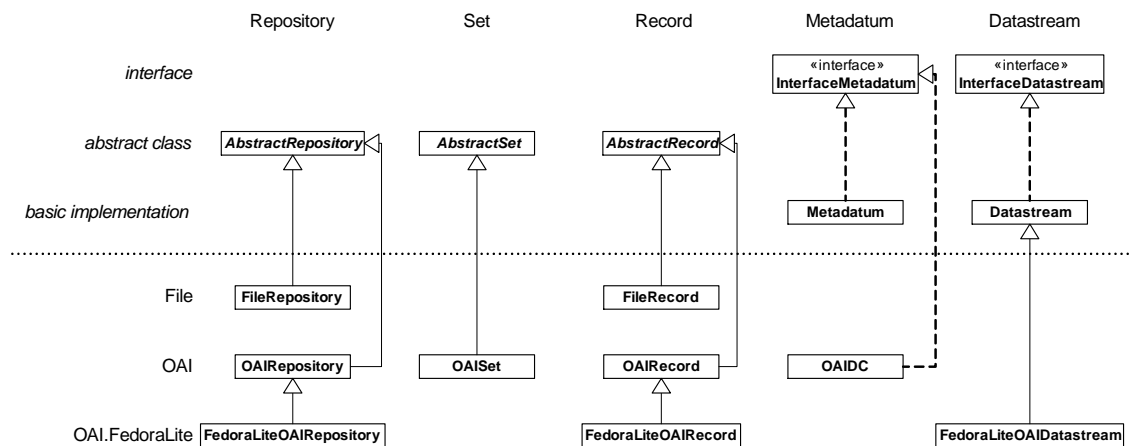


Figure 5.2: Inheritance between the the classes of the implemented adapters.

Microsoft SQL Server is a relational database management system (DBMS) and is used to create and manage the database used for persistence of both the adapter instances and harvested metadata.

Lucene.NET is a port to the .NET framework of the Lucene search library developed by the Apache Software Foundation and is used to create and query indexes to perform searches in the record metadata.

Because of the nature of this architecture as a White Box Framework, special attention is given to the description of the actual public methods, attributes of the classes and relationships between them.

Additionally to the framework itself, several applications that use it were developed.

## 5.2 Abstract Classes and Interfaces

The interface used by applications working on the generic model is defined by AbstractRepository, AbstractSet, AbstractRecord, InterfaceMetadatum, InterfaceDatastream. The abstract methods of

these classes are supposed to be implemented by the corresponding class in each adapter. In the diagrams, names in *italic* correspond to abstract classes and methods. The virtual methods can be overridden to change the default behavior.

### 5.2.1 AbstractRepository

*AbstractRepository* serves as the base class for the Repository of an adapter and defines its generic interface.

<i><b>AbstractRepository</b></i>
<i>+ID : long</i> <i>+RepositoryType : Type</i> <i>+SetType : Type</i> <i>+RecordType : Type</i> <i>+LuceneIndexDirRoot: string</i> <i>+isDeleted : bool</i> <i>+isModified : bool</i>
<i>+readRepositories(in connection: SqlConnection) : ArrayList</i> <i>+writeRepository(in connection: SqlConnection) : int</i> <i>+createLuceneIndex(in connection: SqlConnection in callback: Func&lt;int, int, int&gt;) : int</i> <i>+countSets(in connection: SqlConnection) : long</i> <i>+invalidateSetCount()</i> <i>+readSets(in connection: SqlConnection) : ArrayList</i> <i>+countRecords(in connection: SqlConnection) : long</i> <i>+invalidateRecordCount()</i> <i>+readRecords(in connection: SqlConnection in offset : int, in limit : int) : ArrayList</i> <i>+readRecord(in connection: SqlConnection in id : long) : AbstractRecord</i>

Figure 5.3: AbstractRepository class.

The attribute *ID* is the internal identifier of the repository, unique amongst repositories in the framework.

*AbstractRepository* contains the attributes *RepositoryType*, *SetType*, *RecordType* with the type of the repository, set and record respectively. These allow any method of an adapter to know its implementing classes. Therefore it can find out the related classes even in a derived adapter – by knowing the *Type* and using reflection. For example a method defined in the OAI adapter would normally expect the record class to be *OAIRecord*. However when used from the *OAI.FedoraLite* adapter the record class is *FedoraLiteOAIRecord*. By explicitly stating the type and using reflection the inheritance of methods is more robust.

The attribute *LuceneIndexDirRoot* is used to specify the directory of the Lucene index per-repository if needed. More information on this is provided in section 5.5.1 about the *OAIHarvest* application.

The attribute *isModified* is used to see if a call to *writeSet* is needed to save the data for persistence. Since this is independently of the actual implementing class, it simplifies operations such as saving data before closing the working application.

The static method *readRepositories* is the entry point for the framework. It queries the database for available repositories and returns a list. The method parameters can be used to specify only the repositories belonging to a certain adapter – either directly or by derivation.

The methods `countSets` and `countRecords` return the current number of sets and records, respectively. The actual implementation is defined in the adapter. Since this value can be cached for performance, changes in the number might not be correctly updated. Therefore the methods `invalidateSetCount` and `invalidateRecordCount` exist, to force the next call to reevaluate the returning values.

The method `getSets` returns all the sets of the repository.

The method `getRecords` is used to obtain the records by parts, since the number of records is expected to be high. Using the parameters `limit` and `offset` of the method, it is possible to control which records are returned. This is especially useful for the pagination of the results.

## 5.2.2 AbstractSet

`AbstractSet` serves as the base class for the Set of an adapter and defines its generic interface.

<b><i>AbstractSet</i></b>
+ID : long +Repository : AbstractRepository +isModified : bool +isDeleted : bool +writeSet(in connection : SqlConnection) : int

Figure 5.4: AbstractSet class.

The attribute `ID` is the internal identifier of the set, unique in the repository defined by the attribute `Repository`. A set is uniquely identified by the compound `Repository ID` and `Set ID`

The method `writeSet` is used for the persistence of the set. The actual implementation is done by the deriving classes.

## 5.2.3 AbstractRecord

`AbstractRecord` serves as the base class for the Record of an adapter and defines its generic interface.

<b><i>AbstractRecord</i></b>
+ID : long +Repository : AbstractRepository +isModified : bool +isDeleted : bool +writeRecord(in connection : SqlConnection) : int +getLuceneDocument(in connection : SqlConnection) : Lucene.Net.Documents.Document +getMetadata(in connection : SqlConnection) : ArrayList +getDatastreams(in connection : SqlConnection) : ArrayList

Figure 5.5: AbstractRecord class.



The attribute ID is the internal identifier of the record, unique in the repository defined by the attribute Repository. A record is uniquely identified by the compound Repository ID and Record ID

The method `getLuceneDocument` returns a `Lucene.Net.Documents.Document` [Luc08b] and should be overridden by deriving classes. This is an object used by Lucene for to index the metadata of records. The override should start by calling the base class method to obtain the start document and build from there, before returning it.

For example, `OAIRecord` uses the OAI metadata, using the corresponding key of the OAIDC as the field. This way to search for the word "Study" in the title of records it is possible to use the query "title:Study". This allows for finer searches but depend on the metamodel of the record. The default field, `text`, should contain all the text relevant for generic searching.

The method `getMetadata` returns an `ArrayList` with metadatum elements. The actual type of each element depends on the adapter, yet they all implement the `InterfaceMetadatum`, so the attributes and methods of the generic interface exist.

The method `getDatastream` returns an `ArrayList` with datastream elements. Similarly to `getMetadata`, the actual type depends on the adapter, but all datastreams implement the `InterfaceDatastream`.

### 5.2.4 InterfaceMetadatum

`InterfaceMetadatum` serves as the generic interface for the Metadatum of an adapter.

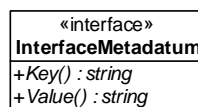


Figure 5.6: InterfaceMetadatum interface.

The interface to Metadatum allows to retrieve a metadata entry as a key-value pair with the `Key` and `Value` attributes respectively.

These attributes are, in the C# language, called properties. It is possible to define the methods `get` and `set` on their value as more than simple retrieval and attribution. More importantly, for this framework, they can be overridden by derived classes.

### 5.2.5 InterfaceDatastream

`InterfaceDatastream` serves as the generic interface for the Datastream of an adapter.

The interface to Datastream allows to retrieve the URL of the datastream and to access it. This is done by calling the method `Execute`, which opens the datastream for use.

As described in Figure 2.4.2.1, URLs are URIs. The `<scheme name>` defines the protocol – that in this context can be any recognized by the operative system [BLMM94], including:

- `file://` — for files accessible by the host computer;

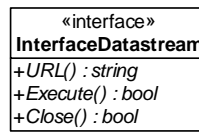


Figure 5.7: InterfaceDatastream interface.

- http:// — for Internet resources accessible using HTTP;
- ftp:// — for Internet resources accessible using FTP.

When the use of the datastream ends, the method Close should be called for cleanup.

## 5.3 Base Classes

The base classes include minimal implementations for the interfaces InterfaceMetadatum and InterfaceDatastream, and the class ArchiveSearch, used to perform searches in repositories.

### 5.3.1 Metadatum

The class Metadatum is a minimal implementation of the interface InterfaceMetadatum.

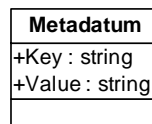


Figure 5.8: Metadatum class.

The class Metadatum simply provides two strings for the key-value pair. No restrictions are placed on either value.

### 5.3.2 Datastream

The class Datastream is a minimal implementation of the interface InterfaceDatastream.

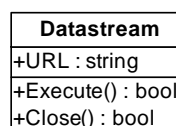


Figure 5.9: Datastream class.

The URL is stored as a simple string. Its execution is performed by launching a new process and having the system open the URL. The actual behavior changes depending on the OS

settings, protocol and file extension. For example, an URL with the protocol "http" would open in the default browser, while the protocol "file" would open the application associated with the file extension. The Close method closes the main window of the new process.

This class is not prepared for persistence.

### 5.3.3 ArchiveSearch

The class ArchiveSearch manages a search query for records in a set of repositories.

ArchiveSearch
-Repositories : ArrayList -Query : string
«constructor» +ArchiveSearch() «constructor» +ArchiveSearch(in repositories : ArrayList, in query : string) +runQuery() : bool +countResults() : int +getResults(in connection : SqlConnection, in offset : int, in limit : int) : ArrayList

Figure 5.10: ArchiveSearch class.

ArchiveSearch is built around the query functionalities of Lucene.NET.

The method runQuery searches the selected repositories for records matching the query. The attributes Repositories and Query can be set either in the call to the constructor or by setting the attributes.

Repositories is an ArrayList containing repository classes that can be of any type as long as it derives from AbstractRepository. Only the indexed records in a repository will be searched.

Query is a string in the Lucene syntax. More information about the syntax of queries in Lucene.NET can be found in the documentation [Luc08a].

After running the query it is possible to get the number and found records.

The method countResults returns the number of results found for the query in the selected repositories.

The method getResults returns the records found matching the query. The actual type of each record returned depend on the repository type it belongs to. For example a record that belongs to a FedoraLiteOAIRepository will be of type FedoraLiteOAIRRecord. This assures that the methods of the record class behave as expected.

The application Repository Searcher (5.5.4) uses this class to search a query in every indexed repository available.

## 5.4 Adapters

The adapters developed were:

- File adapter — for repositories that are a list of URLs for files.

- OAI adapter — for repositories implementing the harvesting protocol OAI-PMH.
- OAI.FedoraLite adapter — for Fedora repositories. It is based on the OAI adapter which performs the harvest of records.

### 5.4.1 Repository Based on File URLs

The File adapter deals with sets of URLs that can be seen as extremely simple repositories. Each URL defines an object whose only datastream is the file itself. The URL can use any protocol, such as HTTP, HTTPS, FTP or the file system. Metadata only includes the URL and a name/alias for the file.

There are two main reasons for this adapter. Firstly, it helped build a more robust and powerful architecture by introducing a simple, but slightly irregular approach to the concept of access to a digital object. Secondly, the Gisa software had a feature consisting in allowing the association of files to a Gisa record. This way a Gisa record can have a list of files on the network that are available for access. Initially only images could be associated and viewed, but by client request PDF documents and other formats were added. However those associations are seen as an *ad hoc* solution that should be phased out in favor of digital objects in a repository system. This adapter can be used as a transition step for those associations – bringing them into this model while a more definitive policy is studied and put into practice.

The internal identification of a record is made using both name and URL, yet these are neither opaque nor persistent.

The classes specific of the File adapter are in under the namespace `ArchiveModel.File`.

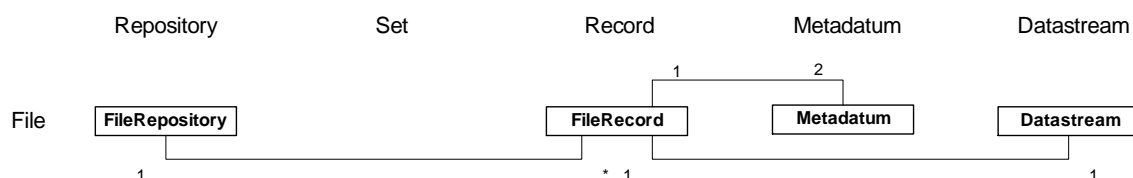


Figure 5.11: Relationship between the classes in the File adapter.

The lack of a Set class and support for sets in this adapter shows that the model can deal with types of repositories where sets are nonexistent.

#### 5.4.1.1 FileRepository

`FileRepository` is the repository class of the File adapter.

The attribute `Name` is a free text field used as an alias of the repository for easier identification.

The attribute `BaseURL` can define the protocol and part of the hierarchy. The URLs of the files in this repository are relative to the `BaseURL` [BLFM05, Fie95]. This helps cope with eventual changes in location of the set of files without having to change each individual URL.

FileRepository
+Name : string
+BaseURL : string

Figure 5.12: FileRepository class.

For example, a possible FileRepository can have as Name the value "Wikimedia Commons" and as BaseURL the value "http://commons.wikimedia.org/wiki/".

#### 5.4.1.2 FileRecord

FileRecord is the record class of the File adapter.

FileRecord
+Name : string
-RelativeURL : string
+getDatastreams(in connection: SqlConnection) : ArrayList
+getMetadata(in connection: SqlConnection) : ArrayList

Figure 5.13: FileRecord class.

The attribute Name is a free text field. It is used as an alternative to the URL to identify the digital object. The attribute Name is considered metadata of the object, being therefore searchable. So, depending on the use, instead of a simple descriptive alias one could use an external identification code.

The attribute RelativeURL has the remaining part of the URL for the individual record. This URL is to be resolved relative to the BaseURL of the FileRepository [BLFM05, 5.2. Relative Resolution].

The method getDatastreams returns exactly one datastream, of the type Datastream. This contains the retrieval URL for the object.

The method getMetadata returns exactly two metadata, of the type Metadatum – one with the value of the Name, the other with the datastream URL.

Following the previous example, a FileRecord could have as Name "Château d'If in Marseille" and RelativeURL "Image:ChateaudIf1900.jpg". This record would have one Datastream with <http://commons.wikimedia.org/wiki/Image:ChateaudIf1900.jpg> as the URL, and two metadata entries, the name and the URL of the datastream.

#### 5.4.2 Repository Based on the OAI-PMH

The OAI adapter uses the OAI Protocol for Metadata Harvesting (OAI-PMH) 2.3.2.1 to harvest information about the repository itself, the sets, digital object records and their metadata.

## Implementation of the Interoperability Framework

The OAI-PMH is a standard protocol implemented for all repository systems studied. This adapter can serve as the base for more specific repositories as demonstrated by the OAI.FedoraLite adapter. [5.4.3](#)

The classes specific of the OAI adapter are under the namespace `ArchiveModel.OAI`.

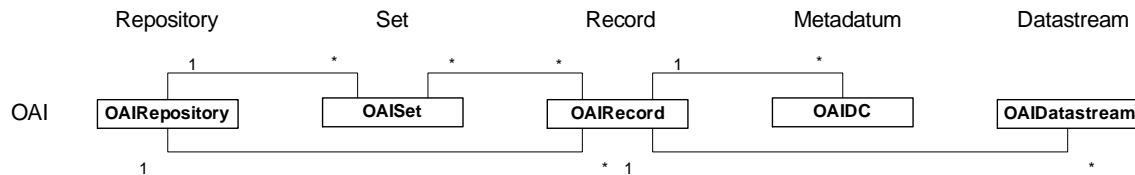


Figure 5.14: Relationship between the classes in the OAI adapter.

The internal identification of records is made using the OAI Identifier.

The OAI adapter is prepared for Unqualified DC metadata only. The presence of this metadata format is mandatory by the OAI-PMH protocol and is the only one guaranteed to be present. This format serves well for free text search or by coarse fields. However, other features such as finer precision search, metadata presentation or crosswalking would be better accomplished using other metadata models.

In the context of the Gisa application, the metadata of documents follow the ISAD(G) specifications ([2.5.2.1](#)). Should the Gisa application import metadata from the digital archives, then harvesting the metadata in EAD format ([2.5.2.2](#)), would allow to crosswalk it to Gisa with better precision. This could be performed by a new adapter derived from this one that, firstly would check for the availability of MetadataPrefix and then either harvest data in all available formats or use a preference list. A metadata registry framework could be used to orchestrate the management of metadata in several formats.

### 5.4.2.1 OAIRepository

OAIRepository is the repository class of the OAI adapter.

OAIRepository
+Name : string
+BaseURL : string
+Harvester : OAIHarvester
+HarvestIdentity(in connection: SqlConnection) : int
+HarvestSets(in connection: SqlConnection in callback: Func<int, int, int>) : int
+HarvestRecords(in connection: SqlConnection in callback: Func<int, int, int>) : int

Figure 5.15: OAIRepository class.

The only information required for the creation of a OAI adapter to a repository is the URL that replies to OAI-PMH requests. This value is in the attribute `BaseURL`.

The method `harvestIdentity` uses the `Identify` verb of the OAI-PMH [[LdSNW02b](#), 4.2 Identify] to automatically populate or update the remaining information attributes. These include

the `Name` attribute and `ProtocolVersion`; `EarliestDatestamp`; `DeletedRecord`; `Granularity`; `AdminEmail`; `Compression`; `Description`.

The `Description` attribute might contain further XML data further describing the repository. This data is kept unprocessed by the OAI adapter.

The method `harvestSets` uses the `ListSets` verb of OAI-PMH [LdSNW02b, 4.6 ListSets] to get the list of sets. This must be done before calling `harvestRecords` or the sets might not exist for the association between records and sets. The `callback` parameter received should be a function that is called by the method to update a progress counter, or null if this isn't needed.

The method `harvestRecords` uses the `ListRecords` verb of OAI-PMH [LdSNW02b, 4.5 ListRecords] to harvest records and their metadata. The `callback` parameter works like in `harvestSets`. There is an overloaded method for finer control of the harvesting. The behavior of this method is shown in the activity diagram 5.25.

This method only retrieves the records for a single call with the `ListRecords` verb which is a part of the total number of records. Therefore it should be called several times, until all records have been harvested has shown in Figure 5.24.

Following harvest to the records will use latest harvest date as the start point. This way an incremental harvesting is performed for new records only.

This class contains additional attributes retaining the state of harvesting of records. This allows keeping track of the current status, incremental harvesting, and resumption of an incomplete harvest if the token has not passed its expiration date.

Harvesting is a relatively lengthy process, so it should be done in a working thread or process, separate from the application GUI.

### 5.4.2.2 OAISet

`OAISet` is the set class of the OAI adapter.

OAISet
+OAIdentifier : string
+Name : string
+Description : string
«constructor» +OAISet(in repository : OAIRepository, in set : OAISet)

Figure 5.16: OAISet class.

The constructor of `OAISet` creates an object using the data received from the harvest. The attributes it fills are used for identification and description of the set.

Since the OAI protocol defines a flat set structure, subsets of a set cannot be inferred without a deeper analysis.

### 5.4.2.3 OAISetRecord

`OAISetRecord` is the record class of the OAI adapter.

OAIRecord
+OAIdentifier : string
+Datestamp : string
«constructor» +OAIRecord(in repository: OAIRepository, in set: OAIHarvestedRecord)

Figure 5.17: OAIRecord class.

The constructor of OAIRecord uses the data received from the harvest to create the record, associate the record to the sets it belong and create the OAIDC metadata. During the harvest of a record the metadata fields are indexed for search. Existing records have their index data updated.

The OAI protocol says nothing about the actual datastream contents of a digital objects, therefore these cannot be accessed with it.

The method `getDatastreams` returns a possibly empty set of datastreams, of the type `Datastream`. These are created by detecting URLs in the metadata fields of the record. Detection is made in each metadatum, by checking if its `Value` starts with either `"http://"`, `"https://"`, `"ftp://"` or `"www."`.

This is an *ad hoc* improvisation upon the realization that many repositories distribute URLs to the digital object front page or content itself. A study of the OAI-PMH protocol in 2005 showed that 73% of the records in the studied repositories had such URLs [MNZL06]. Additional entries could be found if searches included HDL and DOI identifiers. However, this policy changes across repositories so the existence of such URLs and that they provide a link related to the digital object cannot be guaranteed by this approach.

#### 5.4.2.4 OAIDC

OAIDC is the metadatum class of the OAI adapter.

OAIDC
+Key : string
+Value : string

Figure 5.18: OAIDC class.

The metadata format used by this adapter is Unqualified Dublin Core 2.5.2.4. While there are no restrictions to the `Value`, the `Key` can only be one of the 15 values defined in the DC specification: Contributor, Coverage, Creator, Date, Description, Format, Identifier, Language, Publisher, Relation, Rights, Source, Subject, Title, Type.

### 5.4.3 Repository Based on the OAI-PMH and Fedora Access API Lite

The OAI adapter can harvest sets, records and its metadata. However, due to the lack of a common protocol concerning datastreams, these cannot be accessed in an orderly manner. This adapter



allows to access the datastreams of a digital object in a Fedora repository. To do that, the adapter uses one of the APIs provided – the Fedora Access API Lite (API-A-Lite) which uses a REST-like architecture [Com08, Fedora APIs].

The classes specific of the `OAI.FedoraLite` adapter, shown in Figure 5.19, are under the namespace `ArchiveModel.OAI.FedoraLite`.

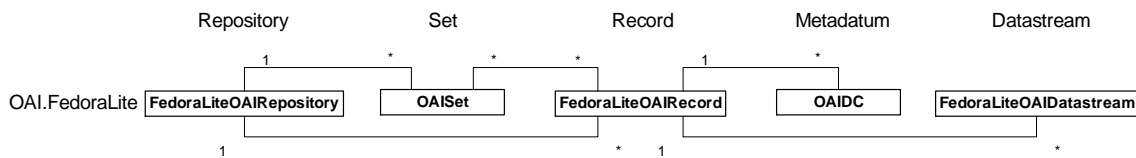


Figure 5.19: Relationship between the classes in the `OAI.FedoraLite` adapter.

The internal identification of records is made using the OAI Identifier. Additionally, the Fedora PID is also used. The PID can be found from the OAI Identifier.

An important aspect of this adapter is that it derives from the OAI adapter. The record and set harvesting is done in the implementation of the parent. Only specificities related to the Fedora repository and its API are present. This validates the inheritance model of the White Box Framework architecture.

Since the `ArchiveModel.OAI.FedoraLite` classes derive from `ArchiveModel.OAI`, then applications working with the more generic adapter will still work for the more specific one. This exemplifies the advantage of following the OO model of inheritance – deriving classes to add new features in a well-planned manner will keep the compatibility between them.

Alternatively to the REST-like API, the SOAP API of Fedora could have been used for the same effect. The SOAP API is more powerful but more complex so, since all the methods required were offered by the REST-like API, this one was used instead.

This adapter works up to version 2.2.1 of Fedora repositories. Starting from version 3 differences in the REST API were introduced that require a new adapter – either by derivation of this one or another set of classes.

### 5.4.3.1 FedoraLiteOAIRepository

`FedoraLiteOAIRepository` is the repository class of the `OAI.FedoraLite` adapter.

FedoraLiteOAIRepository
+FedoraBaseURL: string
+PIDRegex: string
+DefaultPIDRegex: string = @"^oai:[^]*:(?<pid>.*)\$"

Figure 5.20: `FedoraLiteOAIRepository` class.

The common attributes between this and the `OAIRepository` class are persisted using the same database table – only the specific attributes of the adapter are stored in a new table, showing that the inheritance model also reflects in the database.

The `FedoraBaseURL` attribute defines the base URL of the Fedora repository. This is used in the construction of URLs to call its APIs.

The `PIDRegex` attribute defined per-repository is a regular expression that, when run on the OAI Identifier of a record, returns the Fedora PID portion of it. The default value is stored in the static constant `DefaultPIDRegex`:

```
^oai:[^:]*:(?<pid>.*)$
```

This regular expression works for the default naming for OAI Identifier. Taking as an example the identifier `oai:example.org:demo:8` given to a record in the OAI protocol. To use it with the Fedora API the PID `demo:8` is needed. So obtain it, the `<pid>` group in the regular expression is captured.

Regular expressions are a powerful and flexible method to process text [Mer01]. In the case naming is changed in a specific Fedora repository, the default value of the `PIDRegex` attribute might need to be changed. This shows how, by defining the regular expression as an attribute of the repository object, the class might cope with simple changes in behavior without requiring the derivation of a new class.

### 5.4.3.2 FedoraLiteOAIRecord

`FedoraLiteOAIRecord` is the record class of the `OAI.FedoraLite` adapter.

<b>FedoraLiteOAIRecord</b>
+PID : string
+getDatastreams(in connection: SqlConnection) : ArrayList

Figure 5.21: `FedoraLiteOAIRecord` class.

This class overrides `getDatastreams()` of `OAIRecord`. The list of available datastreams for the digital object is obtained using the method `listDatastreams` [Com08, API-A-Lite `listDatastreams`] of the Fedora Lite Access API. This method is called passing the parameter `xml=true` as to receive a XML document. The XML is then parsed to obtain the attributes for the datastreams and a list of `FedoraLiteOAIDatastream` objects are created.

The list of datastreams of a digital object is obtained *on the fly* upon the first call to the `getDatastreams()` method of the object. This assures the freshness of the results. It also causes the limitation that one cannot enquire the available datastreams if the Fedora server is down – and, ironically, the datastreams are unavailable too.

<b>FedoraLiteOAIDatastream</b>
+URL : string
+ItemID : string
+Label : string
+MIME : string

Figure 5.22: FedoraLiteOAIDatastream class.

### 5.4.3.3 FedoraLiteOAIDatastream

FedoraLiteOAIDatastream is the datastream class of the `OAI.FedoraLite` adapter.

The datastream itself is accessed using the method `getDisseminationDatastream` [Com08, API-A-Lite `getDatastreamDissemination`] of the Fedora Lite Access API. By using a REST-like API the access is made using a URL. Therefore we can derive this class from the basic implementation of the Datastream class, passing the adequate URL.

The URL to access the datastream is constructed by concatenating: the Fedora repository base URL; the `get` command of the API; the record PID; the item ID of the datastream (`dsID`). The URL returns the datastream as a mime-typed stream.

The following attributes are present in the Fedora implementation of the datastream class.

- URL – the constructed URL to access the datastream.
- Item ID – the identifier of the datastream (also referred to as `dsID`).

The Item ID is an alphanumeric string, unique in the scope of the object. As such it is used to construct the URL to access the datastream.

- Label – short description of the datastream. The labels of the datastreams are metadata of the digital object, so forethought must be given to both the value and use given.
- MIME type – the internet media type value of the datastream itself, used to identify the content type.

For example the URL `http://localhost:8080/fedora/get/demo:8/DC` would return the Dublin Core metadata as a XML document for the object `demo:8`. The MIME type value for this datastream is `application/xml`. Every digital object in a Fedora repository has a datastream with the `dsID` `DC` containing Dublin Core metadata in XML.

## 5.5 Applications

Several applications were created for the development and testing of the interoperability framework.

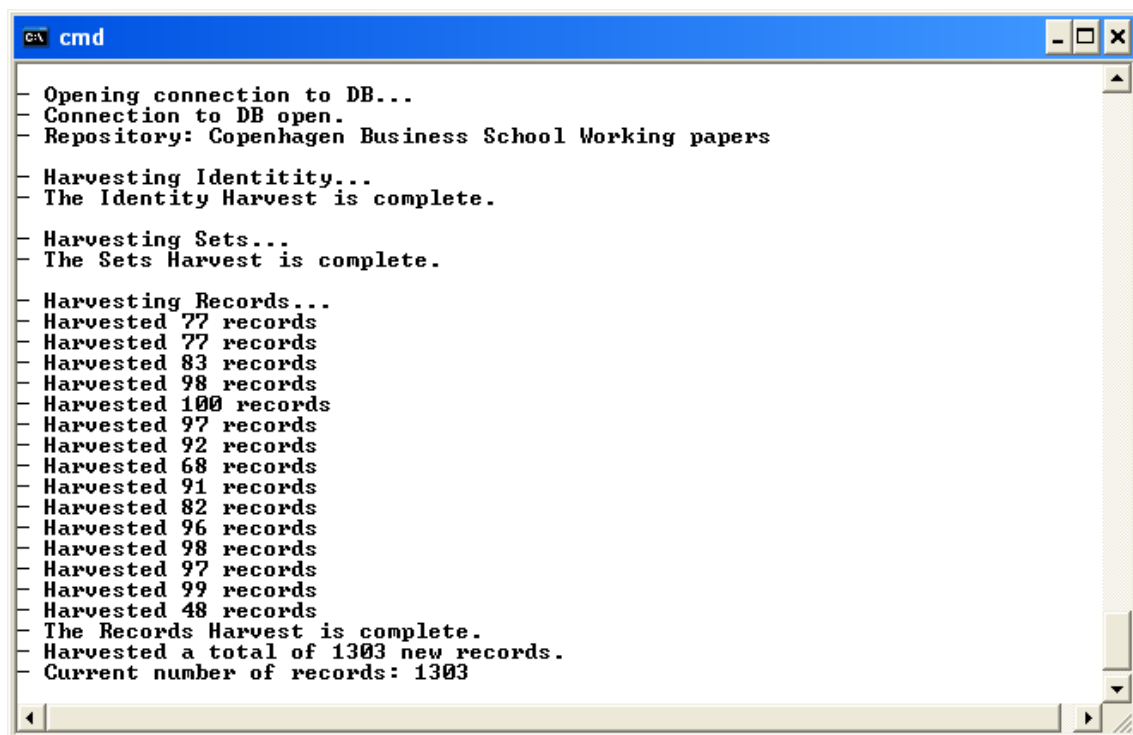
These applications are described in this section and are: `OAIHarvest`; `File Adapter GUI`; `Repository Browser`; and `Repository Searcher`.

### 5.5.1 OAIHarvest

OAIHarvest is a command line application. used to populate the records of a repository that uses the OAI adapter or a derived adapter. It uses the OAI-PMH protocol to make incremental updates when changes occur in the repository. Only changed records are passed by protocol and as such to the adapter.

It was designed considering that it is executed by applications – either cron jobs/Scheduled Tasks updating the records at regular intervals or an update function in an application integrating the interoperability framework. Another advantage of having a separate application perform the updates, is that running in a new process with independent code is a simpler approach compared to multithreading. The persistence database and Lucene indexes are used as common ground to share the data harvested

The OAIHarvest application makes extensive use of the OAI adapter of the framework.



```

cmd
- Opening connection to DB...
- Connection to DB open.
- Repository: Copenhagen Business School Working papers
- Harvesting Identity...
- The Identity Harvest is complete.
- Harvesting Sets...
- The Sets Harvest is complete.
- Harvesting Records...
- Harvested 77 records
- Harvested 77 records
- Harvested 83 records
- Harvested 98 records
- Harvested 100 records
- Harvested 97 records
- Harvested 92 records
- Harvested 68 records
- Harvested 91 records
- Harvested 82 records
- Harvested 96 records
- Harvested 98 records
- Harvested 97 records
- Harvested 99 records
- Harvested 48 records
- The Records Harvest is complete.
- Harvested a total of 1303 new records.
- Current number of records: 1303

```

Figure 5.23: Screenshot of the application OAIHarvest.

The arguments of the application are:

OAIHarvest <Repository ID> <SQL Server DB Connection String> [Options]

- <Repository ID> is the ID number of the repository to perform the harvest on. The Repository Type must either be OAIRepository or FedoraLiteOAIRepository which derives from it.

## Implementation of the Interoperability Framework

- `<SQL Server DB Connection String>` contains the connection information to be passed to the `System.Data.SqlClient.SqlConnection` constructed. An example of the format is: `"user id=USER; password=PASSWORD; server=SERVER; Trusted.Connection=no; database=DATABASE; connection timeout=30"`.
- `[Options]` are an optional white-space separated list with one or more of the following options:
  - `ignore_date` – Starts the harvesting of records from the beginning, ignoring previous harvests and tokens.
  - `ignore_token` – Starts the harvesting of records from the last complete harvest, ignoring tokens from incomplete harvests.
  - `ignore_token_expiration` – Uses a token remaining from an incomplete records harvest is present even if it is past its expiration date. Note that this might be replied with a error message from the OAI provider.
  - `ignore_indexing` – Skips indexing the records metadata for searches. Full reindexing might be done *a posteriori* by running this application again without this option.

The behavior of the application is shown in the activity diagram in Figure 5.24. The actions taken adapt for our model the guidelines recommended by the specification [LdSNW02a, 2.2. Guidelines for Harvester Implementers, 8. Harvesting all the Metadata from a Repository].

Details of the "Harvest Records" action, performed by the method `HarvestRecords` of the class `OAIRepository`, are shown in the activity diagram in Figure 5.25. It gives particular attention to how the options affect the starting point of the harvesting – either use the resumption token, use date of the last complete harvesting or start a new harvesting from the very beginning. The indexing of records by Lucene.NET is also taken into consideration – either incrementally index the new records or perform a full reindexing of all the records.

The date granularity supported by the repository influences the update of the records – while a granularity to the second will only update new records from the second the last harvest started, a granularity to the day will update new records from the current day.

Another important aspect is that the index of records is saved in a directory, defined per repository. By default it is a directory named with the repository ID, relative to the working directory. The directory path can be redefined by changing the attribute `LuceneIndexDirRoot`. This way it is possible to have a finer control over the location of these indexes. For example it is possible to group the indexes of a certain type of repository for specific maintenance purposes. Even if the default behavior is kept, attention must be given to the working directory the application is called in.

Only one instance of the application `OAIHarvest` can be run at a time for each repository ID – this is assured by the file lock in the indexing directory. This way concurrent instances of the harvester working on the same data and OAI Provider are avoided. Otherwise data might be

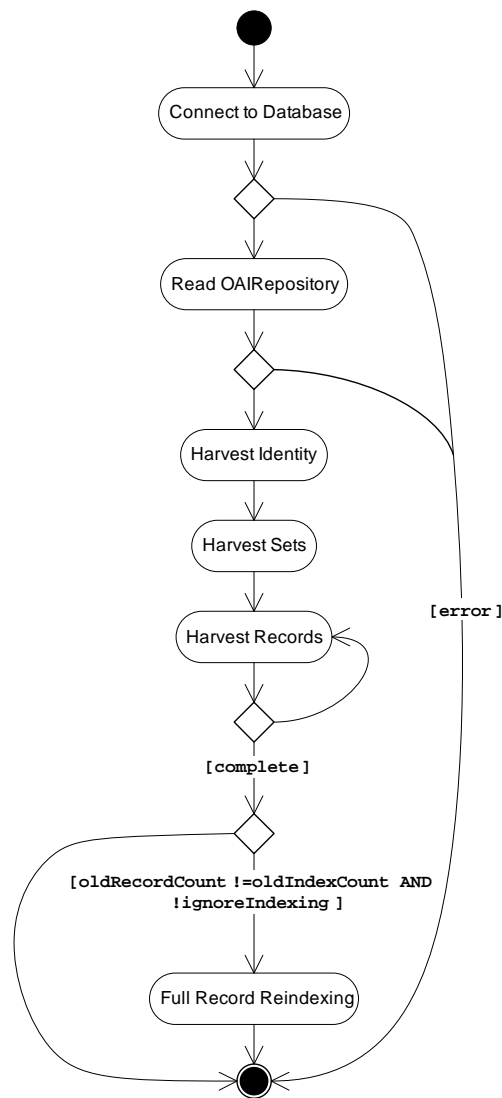


Figure 5.24: Activity diagram of the application OAIHarvest.

harvested twice at the same time, with no advantages and causing an unnecessary load on both the OAI provider and harvester.

A note on some repositories that showed irregular behavior and sent invalid XML, interrupting the harvest. To be valid, a XML document must be well formed and can only contain valid characters. [BPSM<sup>+</sup>06, 2.2 Characters] These are given by the expression:

#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]

DSpace repositories in particular showed this behavior. The manual itself refers to this problem [DSp08, OAI-PMH Data Provider], quoting it:

“the current simple DC implementation (org.dspace.app.oai.OAIDCCrosswalk) does not currently strip out any invalid XML characters that may be lying around in the

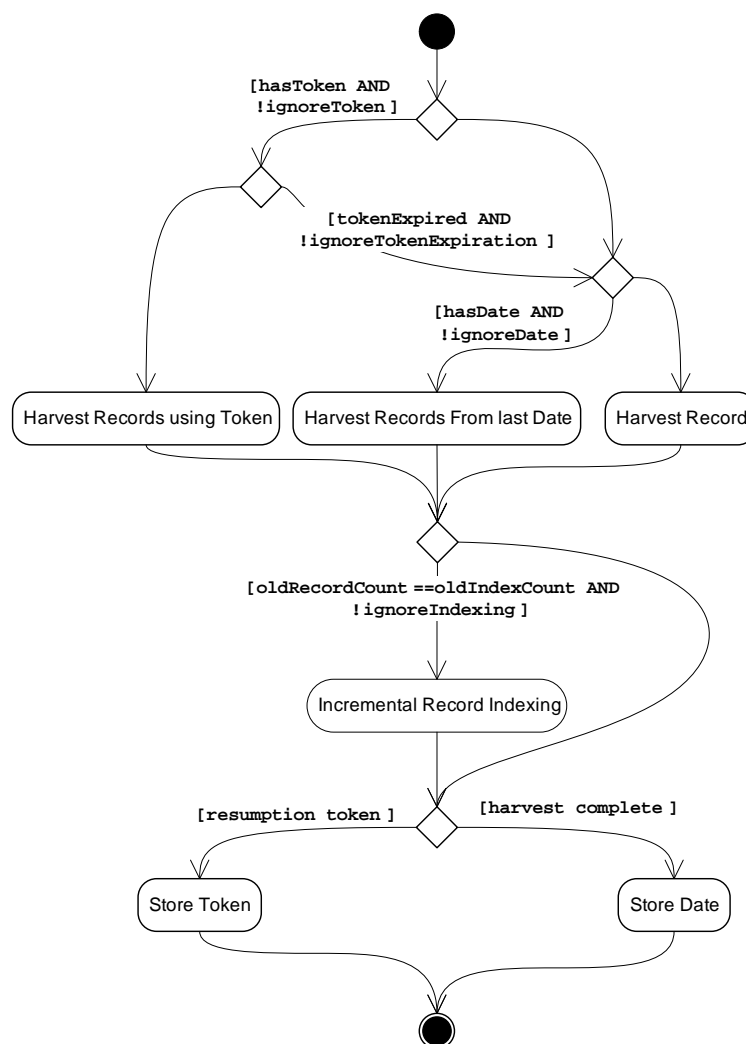


Figure 5.25: Activity diagram of the method `HarvestRecords` of the class `OAIRepository`.

data. If your database contains a DC value with, for example, some ASCII control codes (form feed etc.) this may cause OAI harvesters problems. This should rarely occur, however.”

This is a fault of the repository system that can be solved on its side by cleaning the metadata before constructing the XML documents.

### 5.5.2 File Adapter GUI

File Adapter GUI is a .NET application for the management of repositories that use the File Adapter.

Two screenshots of the main window are shown in [Figure 5.26](#)

The controls displayed are the same in both screenshots and are, ordered by left-right, top-down: button to add a new File repository; dropdown list of current File repositories; button to

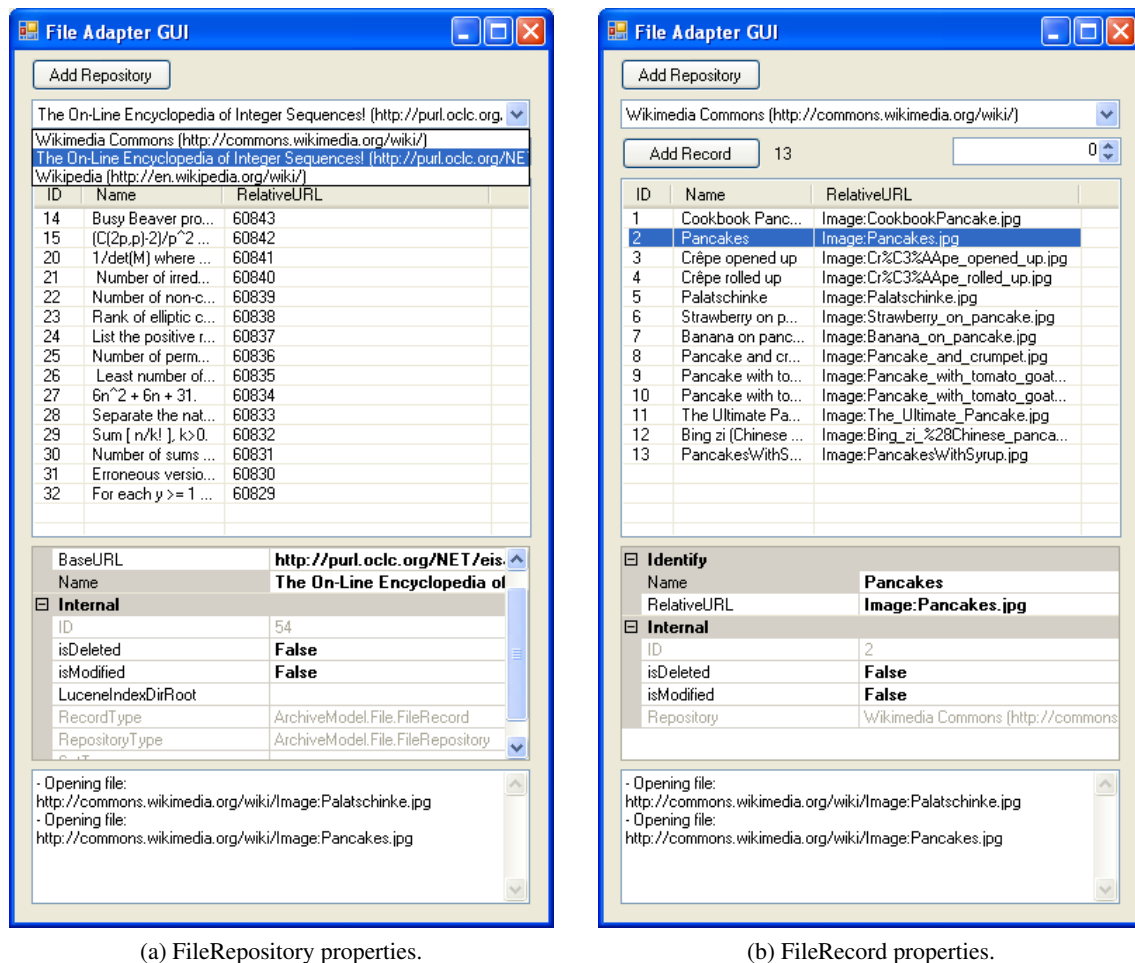


Figure 5.26: Screenshots of the application File Adapter GUI.

add a new File record to the repository; page number control; list of records; the properties control for the selected item; and the message window.

Some notes about the behavior of the application:

- The list of records shows a maximum of 15 records, and has a page number control, controlling which part of the full collection of records is shown.
- Double-clicking a record will call the `Execute` method of its single datastream (5.2.5).
- When either a repository or a record are selected, the corresponding object can be analyzed and edited in the properties control.
- The message window is used to display informative and error messages, to help in the development and debugging of both the application and framework.

The reason for its development was to test the File adapter, and the data model for a specific adapter.



## Implementation of the Interoperability Framework

The main features tested are related with the File adapter: listing of repositories using a specific adapter; creation of new File repositories; creation of new records on those repositories; edition of properties of both File repositories and adapters; access to datastreams.

### 5.5.3 Repository Browser

Repository Browser is a .NET application that is used to browse through any repository of the framework. Using any of the current adapters – File, OAI, OAI.FedoraLite – and, since it uses the generic interface of the adapter model, any further developed adapter would work in this application.

The main window is shown in Figure 5.27.

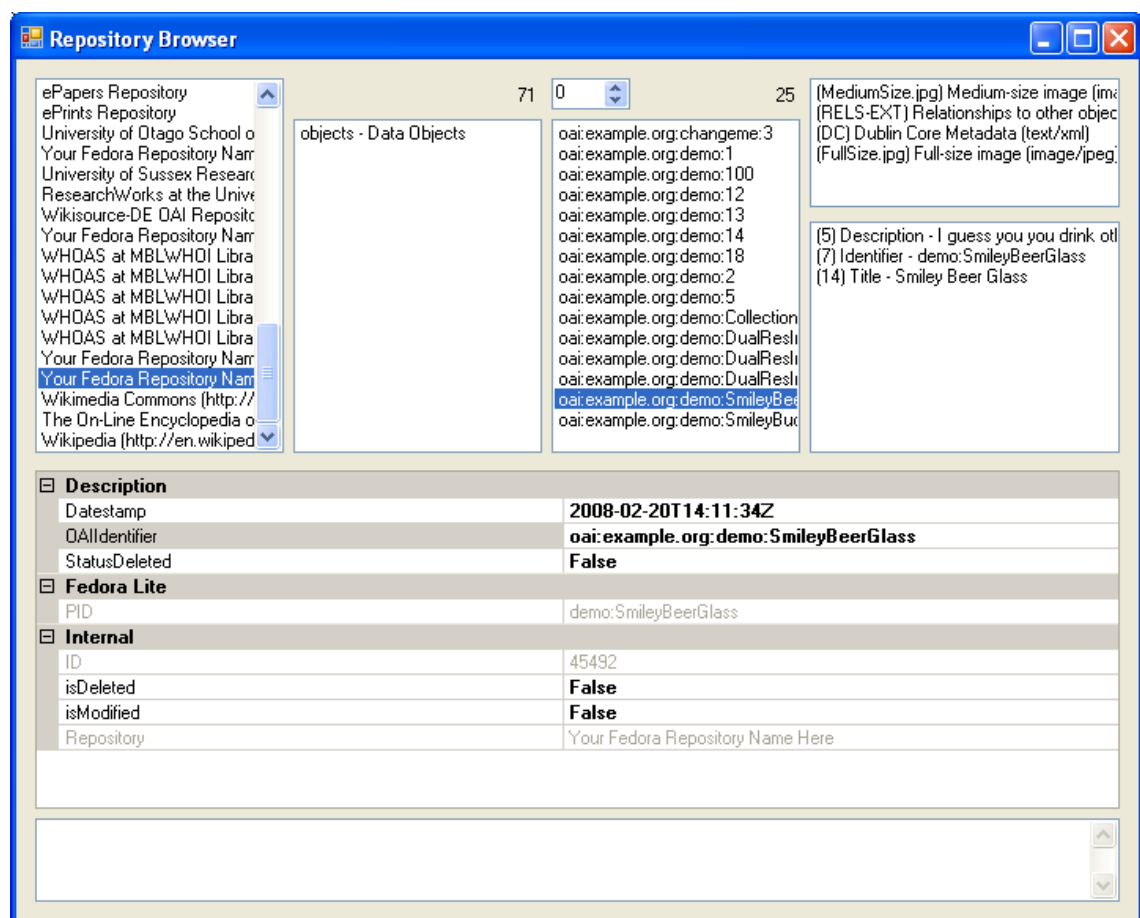


Figure 5.27: Screenshot of the application Repository Browser.

The controls displayed are, ordered by left-right, top-down: the list of repositories; the count and list of sets; the count and list of records; the list of datastreams; the list of metadata; the properties control for the selected item; and the message window.

Some notes about the behavior of the application:

- When a record is selected, only the sets the record belongs to are seen in the list of sets.

- Double-clicking a datastream will call its `Execute` method (5.2.5).
- When an item of a list is selected – be it a repository, a set, a record, a datastream or a metadatum – the corresponding object can be analyzed in the properties control.
- The functions of both the page number control and message window are the same as the File Adapter GUI.

The reason for its development was to test the generic interface, data model and browsing through the available objects.

The main features tested are related with the generic interface of the framework: listing of the known repositories; for each repository listing of their sets, records with pagination; for each record, viewing of the corresponding metadata in key-value format and datastreams; access to datastreams.

### 5.5.4 Repository Searcher

Repository Searcher is a very simple command-line application. It asks for the user to input text string, for a query in the metadata of the records. Then, using the class `ArchiveSearch`, it searches in all indexed repositories for records matching the query. Finally it returns the matches and lists them in sets of 15 records.

The reason for its development is the testing of the class `ArchiveSearch` (5.3.3) and the Lucene indexes created by an adapter.

The main features tested are related with searches: validity of the indexes; simultaneous searches over multiple repositories; query processing; pagination of results; matching records returned on their corresponding type – a match on a record using an OAI adapter will return a `OAIRecord`.

## 5.6 Summary

This section demonstrates the details of implementation of the interoperability framework. Abstract classes and interfaces for the generic interface and data model. The actual implementation and business logic is done by derived classes in a series of adapters. OO concepts of inheritance and polymorphism coordinate the relationships between classes. Three adapters and their corresponding classes are described.

The developed applications are also described – the OAI Harvester, developed for the use with the OAI adapter, and additional test applications, showing the correct functioning of the model and implementation.

## Chapter 6

# Conclusions and Future Work

This chapter summarizes the project, discusses the satisfaction of its objectives and presents possible directions for future work.

### 6.1 Discussion

The project ran according to the initial plan. In the first phase, a list of requirements for a repository in a digital archives was compiled, then the selected open source repository systems (Fedora, DSpace and EPrints) were analyzed on these requirements. A special attention was given to the interoperability aspects, such as exposed APIs and standard protocols implemented by the repository systems. In this part of the study, the commercial repository system DigiTool was included. This study allowed to discover the common aspects and divergences between repository systems, setting the stage for the second phase, the development of the interoperability framework. Its architecture was developed and refined in parallel with the implementation of both the framework and the related applications using it. The framework is based on the concept of adapters, components containing the business logic to interact with repositories and expose the results in a generic model. These adapters have hierarchical relationships, with a common generic interface at the top, from which less abstract adapters are derived to implement protocols in repositories.

The objectives are fully satisfied. The study, it provides an introduction for the selection and deployment of a repository based on any of the repository systems analyzed. The integration module, it was implemented as an interoperability framework that can be used by Gisa and any application which requires communication with repositories of digital archives. The flexibility and extensibility of this framework were aspects that were taken into consideration since the beginning and during the whole development.

The identification scheme for digital objects is univocal, homogenous and independent of the repository itself. This allows the associations with digital objects to reference them always in the same way, simplifying the creation of such associations. Relatively to the search functionalities

in the contents of a digital archive, the framework can search through the metadata contents of several different repositories simultaneously and return the results in an homogenous manner.

### 6.2 Future Work

While the current implementation of the framework satisfies the objectives, it still has potential for further development. It can grow in three dimensions. Firstly, by the creation of adapters to be used for other repository systems besides Fedora: such as the ones studied, DSpace, EPrints and DigiTool. Secondly, by using additional standard protocols and covering additional functionalities, like using SWORD for the ingestion by the repository of digital objects and OpenURL for the exportation of known digital objects. Thirdly, by refining the framework, which could be done on several aspects, such as making persistence independent of a specific DBMS and encapsulating common functionalities in classes to be instantiated rather than derived. This last aspect is the common direction taken by maturing White Box Frameworks, which starts by encapsulating classes, becoming closer to a Black Box Framework. This kind of framework is easier to use as it does not require as much knowledge about how its internal implementation works.

Another step in the continuation of this work is to follow the integration of the interoperability framework with Gisa and other applications. This will highlight any existing issues, and might give further insights to the development of the framework.

# References

- [AFL08] J. Allinson, S. François, and S. Lewis. SWORD: Simple Web-service Offering Repository Deposit. *Ariadne Magazine*, April 2008. <http://www.ariadne.ac.uk/issue54/allinson-et-al/>.
- [arX07] Understanding the arXiv identifier, April 2007. [http://arxiv.org/help/arxiv\\_identifier](http://arxiv.org/help/arxiv_identifier).
- [Bal06] Alex Ball. Briefing Paper: the OAIS Reference Model, February 2006. <http://homes.ukoln.ac.uk/~ab318/eprints/oaisBriefing.pdf>.
- [BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. Technical Report 3986, RFC, January 2005. <http://www.ietf.org/rfc/rfc3986.txt>.
- [BLMM94] T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators (URL). Technical Report 1738, RFC, December 1994. <http://www.ietf.org/rfc/rfc1738.txt>.
- [BP01] C. Blanchi and J. Petrone. Distributed Interoperable Metadata Registry. *D-Lib Magazine*, 7(12):1082–9873, 2001. <http://www.dlib.org/dlib/december01/blanchi/12blanchi.html>.
- [BPSM<sup>+</sup>06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition). *W3C Recommendation*, August 2006. <http://www.w3.org/TR/REC-xml/>.
- [BVdS05] J. Bekaert and H. Van de Sompel. Access Interfaces for Open Archival Information Systems based on the OAI-PMH and the OpenURL Framework for Context-Sensitive Services. *Arxiv preprint cs.DL/0509090*, 2005. <http://arxiv.org/abs/cs/0509090>.
- [Col06] Jennifer Colvin. Inside CDL: Digital Objects Glossary, May 2006. <http://www.cdlib.org/inside/diglib/glossary/?field=glossary&action=search&query=oac>.
- [Com08] Fedora Commons Community. Fedora Commons, 2008. [http://www.fedora.info/wiki/index.php/Main\\_Page](http://www.fedora.info/wiki/index.php/Main_Page).
- [Coy06] Karen Coyle. Identifiers: Unique, persistent, global. *The Journal of Academic Librarianship*, 32:428–431, July 2006. <http://www.kcoyle.net/jal-32-4.html>.
- [DC] Dublin Core Metadata Initiative (DCMI). <http://dublincore.org/>.

## REFERENCES

- [DDK07] Lorrie Dong, Megan Durden, and Sarah Kim. Wesker Project Final Report - Appendix E: Batch Ingest Process, May 2007. <https://pacer.ischool.utexas.edu/handle/2081/8870>.
- [Dig] Managing and Showcasing Digital Collections and Institutional Repositories). <http://www.exlibrisgroup.com/category/DigiToolOverview>.
- [DOI] The Digital Object Identifier System. <http://www.doi.org/>.
- [DSp] DSpace). <http://www.fedora-commons.org/>.
- [DSp08] *DSpace Manual (Version 1.5)*, May 2008. [http://www.dspace.org/index.php?option=com\\_content&task=view&id=151](http://www.dspace.org/index.php?option=com_content&task=view&id=151).
- [ea05] Eric Jansson et al. Elated: a general-purpose web-based client for the Fedora Repository System, 2005. <http://elated.sourceforge.net/index.html>.
- [EAD] EAD: Encoded Archival Description Version 2002 Official Site (EAD Official Site, Library of Congress). <http://www.loc.gov/ead/>.
- [EPr] Open Access and Institutional Repositories with EPrints). <http://www.eprints.org/>.
- [EPr07] *EPrints Manual (Version 3.0)*, 2007. <http://wiki.eprints.org/w/Documentation>.
- [EPr08] EPrints Wiki, June 2008. <http://wiki.eprints.org/>.
- [eSc08] eSciDoc. *Fedora Performance and Scalability Wiki Online*, 2008. <http://fedora.fiz-karlsruhe.de/docs/Wiki.jsp?page=Main>.
- [ExL05] ExLibris. DigiTool and ALEPH Q&A. *The Ex Librian*, 2(1), 2005. <http://www.exlibrisgroup.com/files/Newsletter/ExLibrianNov2007/April2005newsletter.pdf>.
- [ExL06] ExLibris. *The DigiTool X-Server (DigiTool Version 3)*, January 2006.
- [ExL08] ExLibris. *Repository Web Services (DigiTool Version 3)*, February 2008.
- [FCR<sup>+</sup>07] M. Ferreira, R. Castro, J.C. Ramalho, F. Barbedo, L. Corujo, and L. Faria. RODA: Repositório de Objectos Digitais Autênticos, 2007. <http://repositorium.sdum.uminho.pt/handle/1822/6330>.
- [Fed] Fedora Commons). <http://www.fedora-commons.org/>.
- [Fed05] *Fedora System Documentation (Fedora Version 2.0)*, 2005. <http://fedora.info/download/2.0/userdocs/>.
- [Fed07a] *Fedora Installation and Configuration Guide (Fedora Version 2.2.1)*, 2007. <http://www.fedora-commons.org/documentation/2.2.1/userdocs/distribution/installation.html>.
- [Fed07b] *Fedora System Documentation (Fedora Version 2.2.1)*, 2007. <http://www.fedora-commons.org/documentation/2.2.1/userdocs/>.

## REFERENCES

- [Fed07c] *Fedora Directory Ingest Service Documentation (Fedora Version 2.2)*, 2007. <http://www.fedora.info/download/2.2/services/diringest/doc/index.html>.
- [Fed07d] *Fedora Generic Search Service Documentation (Fedora Version 2.2)*, 2007. <http://www.fedora.info/download/2.2/services/genericsearch/doc/index.html>.
- [Fed07e] *Fedora OAI Provider Service Documentation (Fedora Version 2.2)*, 2007. <http://www.fedora.info/download/2.2/services/oaiprovider/doc/index.html>.
- [Fed08] *Fedora System Documentation (Fedora Version 3.0 Beta 2)*, 2008. <http://www.fedora-commons.org/documentation/3.0b1/userdocs/>.
- [Fie95] R. Fielding. Relative Uniform Resource Locators. Technical Report 1808, RFC, June 1995. <http://www.ietf.org/rfc/rfc1808.txt>.
- [For08] Foresite - Functional Object Re-use and Exchange: Supporting Information Topology Experiments, June 2008. <http://foresite.cheshire3.org/>.
- [GPL] <http://www.opensource.org/licenses/gpl-2.0.php>. Open Source Initiative OSI - The GPL:Licensing.
- [Gra00] S. Granger. Emulation as a digital preservation strategy. *D-Lib Magazine*, 6(10):1082–9873, 2000.
- [Hag03] K. Hagedorn. OAIster: a “no dead ends” OAI service provider. *Library Hi Tech*, 21(2):170–181, 2003. [http://chaos.vtls.com/oai\\_docs/OAI\\_OAIster.pdf](http://chaos.vtls.com/oai_docs/OAI_OAIster.pdf).
- [HDL] The Handle System. <http://www.handle.net/>.
- [HN06] Terry L. Harrison and Michael L. Nelson. Just-in-time recovery of missing web pages. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 145–156, New York, NY, USA, 2006. ACM. <http://doi.acm.org/10.1145/1149941.1149971>.
- [Inf02] Infonortics. Peer, Meta and Distributed Search. Search Engines Meeting, 2002. <http://www.searchtools.com/slides/searchengines2002/>.
- [ISA99] ISAD(G) General International Standard Archival, September 1999. [http://www.icacds.org.uk/eng/ISAD\(G\).pdf](http://www.icacds.org.uk/eng/ISAD(G).pdf).
- [ISO03] *ISO/IEC 11179-3:2003 - Information technology – Metadata registries (MDR) – Part 3: Registry metamodel and basic attributes*, 2003.
- [KMSN07] M. Klein, F. McCown, J. A. Smith, and M.L. Nelson. How much preservation do I get if I do absolutely nothing?, January 2007.
- [Kun03] J.A. Kunze. Towards electronic persistence using ARK identifiers. *Third ECDL Workshop on Web Archives*, 2003. <http://www.cdlib.org/inside/diglib/ark/arkcdl.pdf>.

## REFERENCES

- [LC06] J. Lu and J. Callan. Full-text federated search of text-based digital libraries in peer-to-peer networks. *Information Retrieval*, 9(4):477–498, 2006. [http://www.cs.cmu.edu/~jielu/Papers/jir\\_jielu\\_p2pir.pdf](http://www.cs.cmu.edu/~jielu/Papers/jir_jielu_p2pir.pdf).
- [LdSJ<sup>+</sup>08a] Carl Lagoze, Herbert Van de Sompel, Pete Johnston, Michael Nelson, Robert Sanderson, and Simeon Warner. *ORE Specification and User Guide*, June 2008. <http://www.openarchives.org/ore/0.9/>.
- [LdSJ<sup>+</sup>08b] Carl Lagoze, Herbert Van de Sompel, Pete Johnston, Michael Nelson, Robert Sanderson, and Simeon Warner. *ORE User Guide - Resource Map Discovery*, June 2008. <http://www.openarchives.org/ore/0.9/discovery.html>.
- [LdSNW02a] Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simeon Warner. Implementation Guidelines for the Open Archives Initiative Protocol for Metadata Harvesting, June 2002. <http://www.openarchives.org/OAI/2.0/guidelines.htm>.
- [LdSNW02b] Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simeon Warner. The Open Archives Initiative Protocol for Metadata Harvesting, June 2002. <http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm>.
- [LMZN01] X. Liu, K. Maly, M. Zubair, and M.L. Nelson. Arc - An OAI Service Provider for Digital Library Federation. *D-Lib Magazine*, 7(4):1–17, 2001. <http://webdoc.sub.gwdg.de/edoc/aw/d-lib/dlib/april01/liu/04liu.html>.
- [Luc08a] *Apache Lucene - Query Parser Syntax*, January 2008. <http://lucene.apache.org/java/docs/queryparsersyntax.html>.
- [Luc08b] Apache Lucene.Net 2.1 Class Library API, 2008. <http://incubator.apache.org/lucene.net/docs/2.1/>.
- [Mer01] Brad Merrill. C# Regular Expressions, January 2001. [http://www.windowsdevcenter.com/pub/a/oreilly/windows/news/csharp\\_0101.html](http://www.windowsdevcenter.com/pub/a/oreilly/windows/news/csharp_0101.html).
- [MET] METS Schema, & Documentation: Metadata Encoding and Transmission Standard (METS). <http://www.loc.gov/standards/mets/mets-schemadocs.html>.
- [MN06] Frank McCown and Michael L. Nelson. Evaluation of crawling policies for a web-repository crawler. In *HYPertext '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 157–168, New York, NY, USA, 2006. ACM. <http://doi.acm.org/10.1145/1149941.1149972>.
- [MNZL06] F. McCown, M.L. Nelson, M. Zubair, and X. Liu. Search Engine Coverage of the OAI-PMH Corpus. *IEEE INTERNET COMPUTING*, pages 66–73, 2006. <http://library.lanl.gov/cgi-bin/getfile?LA-UR-05-9158.pdf>.
- [Moa97] R. Moats. URN Syntax. Technical Report 2141, RFC, May 1997. <http://www.ietf.org/rfc/rfc2141.txt>.
- [MSN06] Frank McCown, Joan A. Smith, and Michael L. Nelson. Lazy preservation: reconstructing websites by crawling the crawlers. In *WIDM '06: Proceedings of the*



## REFERENCES

- 8th annual ACM international workshop on Web information and data management*, pages 67–74, New York, NY, USA, 2006. ACM. <http://doi.acm.org/10.1145/1183550.1183564>.
- [MyS] *MySQL 5.0 Reference Manual - Chapter 16. Replication*. <http://dev.mysql.com/doc/refman/5.0/en/replication.html>.
- [OAI02] Reference model for an open archival information system (oais), January 2002. <http://public.ccsds.org/publications/archive/650x0b1.pdf>.
- [Ope] The OpenURL Framework for Context-Sensitive Services. [http://www.niso.org/kst/reports/standards?step=2&gid=&project\\_key=d5320409c5160be4697dc046613f71b9a773cd9e](http://www.niso.org/kst/reports/standards?step=2&gid=&project_key=d5320409c5160be4697dc046613f71b9a773cd9e).
- [Pow05] Andy Powell. A 'service oriented' view of the JISC Information Environment. *UKOLN*, November 2005. <http://www.ukoln.ac.uk/distributed-systems/jisc-ie/arch/soa/jisc-ie-soa.pdf>.
- [PRE] PREMIS: PREservation Metadata Implementation Strategies [OCLC - Projects]. <http://www.oclc.org/research/projects/pmwg/>.
- [pro07] PILIN project. Identifier Service Guidelines, September 2007. <http://resolver.net.au/hdl/102.100.272/1KKBLPDQH>.
- [PUR] Persistent Uniform Resource Locator. <http://purl.org/>.
- [Roe05] E. Roel. The MOSC Project: Using the OAI-PMH to Bridge Metadata Cultural Differences across Museums, Archives, and Libraries Eulalia Roel. *INFORMATION TECHNOLOGY AND LIBRARIES*, 2005. <http://news.ala.org/ala/lita/litapublications/ital/volume242005/number1march/contentabc/barnettellis.pdf>.
- [RZ06] Mohamed Rafi and Xiaofang Zhou. Fedora-SRB Database integration module, October 2006. <http://www.itee.uq.edu.au/~ereseach/projects/dart/outcomes/FedoraDB.php>.
- [SN08] Joan A. Smith and Michael L. Nelson. Creating Preservation-Ready Web Resources. *D-Lib Magazine*, 14, jan, feb 2008. <http://dlib.org/dlib/january08/smith/01smith.html>.
- [SRU] SRU: Search/Retrieval via URL – SRU, CQL and ZeeRex (Standards, Library of Congress). <http://www.loc.gov/standards/sru/>.
- [Sto06] Larry Stone. Lightweight Network Interface, May 2006. <http://wiki.space.org/LightWeightNetworkInterface>.
- [SWJF96] Keith Shafer, Stuart Weibel, Erik Jul, and Jon Fausey. Introduction to Persistent Uniform Resource Locators. *INET'96*, 1996. <http://purl.oclc.org/docs/inet96.html>.
- [Thi02] K. Thibodeau. Overview of Technological Approaches to Digital Preservation and Challenges in Coming Years. *The State of Digital Preservation: An International Perspective*, 2002.

## REFERENCES

- [VdSBL<sup>+</sup>05] H. Van de Sompel, J. Bekaert, X. Liu, L. Balakireva, and T. Schwander. aDORe: A Modular, Standards-Based Digital Object Repository. *The Computer Journal*, 48(5):514–535, 2005. <http://comjnl.oxfordjournals.org/cgi/content/abstract/48/5/514>.
- [VdSYH03] H. Van de Sompel, J.A. Young, and T.B. Hickey. Using the OAI-PMH... Differently. *D-Lib Magazine*, 9(7/8):1082–9873, 2003. <http://www.dlib.org/dlib/july03/young/07young.html>.
- [W<sup>+</sup>06] R. Wyles et al. Technical Evaluation of selected Open Source Repository Solutions v1.3. *Open Access Repositories in New Zealand (OARINZ) project*, Christchurch Polytechnic Institute of Technology, 13, 2006.
- [Wea05] Belinda Weaver. FEZ development at the University of Queensland Library, October 2005. <http://www.library.uq.edu.au/escholarship/fezdev.pdf>.
- [Wei06] Stuart Weibel. Identifier ideology: Opacity and semantics, February 2006. [http://weibel-lines.typepad.com/weibelines/2006/02/identifier\\_ideo.html](http://weibel-lines.typepad.com/weibelines/2006/02/identifier_ideo.html).
- [Wil08] Andrew Wilson. Significant Properties of Digital Objects. JISC Significant Properties Workshop, British Library, April 2008. <http://www.dpconline.org/docs/events/080407sigpropsWilson.pdf>.

## Appendix A

# Repository Systems Contacts

### A.1 Fedora

- Homepage: <http://www.fedora-commons.org/>
- Wiki: <http://www.fedora.info/wiki/>
- SourceForge: <http://sourceforge.net/projects/fedora-commons/>
- Mailing Lists: 1. [fedora-commons-users](#) 2. [fedora-commons-developers](#)

### A.2 DSpace

- Homepage: <http://www.dspace.org/>
- Wiki: <http://wiki.dspace.org/>
- SourceForge: <http://sourceforge.net/projects/dspace/>
- Mailing Lists: 1. [dspace-general](#) 2. [dspace-tech](#) 3. [dspace-devel](#)
- IRC Channels: 1. [#dspace@irc.freenode.net](#) 2. [#dspace-ukug@irc.freenode.net](#)
- Live Chat – Meebo: <http://widget.meebo.com/mm.swf?OBNiXNAKsR>

### A.3 EPrints

- Homepage: <http://www.eprints.org/>
- Wiki: <http://wiki.eprints.org/>
- Mailing Lists: 1. [eprints\\_tech](#)

### A.4 Digitool

- Homepage: <http://www.exlibrisgroup.com/category/DigiToolOverview>